AN INVESTIGATION OF THE SCHEDULING ASPECTS OF MULTIPROGRAMMING

WALLACE G. WILDER

AN INVESTIGATION
OF THE SCHEDULING ASPECTS
OF MULTIPROGRAMMING

* * * * * * *

Wallace G. Wilder

AN INVESTIGATION

OF THE SCHEDULING ASPECTS

OF MULTIPROGRAMMING

by

Wallace G. Wilder

Lieutenant, United States Navy

Submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE
IN
ENGINEERING ELECTRONICS

United States Naval Postgraduate School
Monterey, California

1964

AN INVESTIGATION

OF THE SCHEDULING ASPECTS

OF MULTIPROGRAMMING

by

Wallace G. Wilder

This work is accepted as fulfilling

the thesis requirements for the degree of

MASTER OF SCIENCE

IN

ENGINEERING ELECTRONICS

from the

United States Naval Postgraduate School

ABSTRACT

An evaluation of several of the various methods available for analyzing multiprogramming scheduling algorithms is conducted. The evaluation consists of an analysis of the capabilities and limitations of the methods and an examination of their effectiveness, when used on three different types of scheduling algorithms. The three types of algorithms are: (1) a single-level round-robin type, (2) a multi-level dynamic priority allocation scheme, and (3) a three-level implicit priority method proposed by the author for general application, including sample-data control use. The results of the evaluation are used in formulating a composite investigation procedure that contains the best features of the various methods of analysis.

The author wishes to express his appreciation to Professor Mitchell L. Cotton of the U. S. Naval Postgraduate School for his assistance and encouragement during this investigation.

TABLE OF CONTENTS

Section	Title	Page
1.0	Introduction	1
2.0	Multiprogramming Background	4
	2.1 Motivations for Use of Time-Sharing	4
	2.2 System Goals	5
	2.3 Scheduling	7
	2.3.1 System Goals as Design Considerations	8
	2.3.2 Overhead Considerations	9
	2.3.3 Scheduler Input Considerations	9
	2.3.4 Cost-Effectiveness Considerations	10
	2.3.5 Summary of General Scheduler Design Considerations	10
3.0	Investigation	11
	3.1 Methods of Investigation	11
	3.1.1 Heuristic Approach	11
	3.1.2 Queuing Theory Approach	12
	3.1.3 Simulation Approach	12
	3.1.4 System Implementation Approach	13
	3.2 Proposed Composite Investigation Procedure	13
	3.2.1 Preliminary Survey	14
	3.2.2 Collection of Statistical Data	14
	3.2.3 Detailed System Study	15
	3.2.4 Small Scale Trial	15
	3.2.5 Full Scale Implementation	15

Section	Title	Page
3.3	Preliminary Survey by Heuristic Analysis	16
3.3.1	Algorithm A	17
3.3.1.1	Operation of Algorithm A	17
3.3.1.2	Effects of Varying Basic Parameters of Algorithm A	19
3.3.1.3	Summary of Analysis of Algorithm A	26
3.3.2	Algorithm B	27
3.3.2.1	Operation of Algorithm B	27
3.3.2.2	Effects of Varying Basic Parameters of Algorithm B	28
3.3.2.3	Summary of Analysis of Algorithm B	30
3.3.3	Algorithm C	32
3.3.3.1	Operation of Algorithm C	36
3.3.3.2	Effects of Varying Basic Parameters of Algorithm C	39
3.3.3.3	Limitations of Algorithm C	41
3.3.3.4	Proposed Changes to Algorithm C	41
3.3.3.5	Summary of Analysis of Algorithm C	42
3.4	Detailed System Study By Simulation	43
3.4.1	Features of The Simulator	43
3.4.1.1	Input Data Features	43
3.4.1.2	Output Data Features	44

Section	Title	Page
	3.4.2 Example of Simulator Operation	46
	3.4.2.1 Input Data	47
	3.4.2.2 Variations of The Algorithm To Be Considered	47
	3.4.2.3 Effects of Early Quantum Termination	49
	3.4.2.4 Effects of Allowing Background Users	54
	3.4.2.5 Effects of Varying Maximum Number of Users	55
	3.4.2.6 Effects of Varying The Rate of Arrival of Service Requests	55
	3.4.3 General Summary of Simulation Analysis	58
4.0	Conclusions and Recommendations	61
	Bibliography	65
	APPENDICES	
I.	SIM - A Multiprogramming System Simulator	66
II.	Program SIM	72

LIST OF ILLUSTRATIONS

Figure		Page
1.	Effects of Variations in Algorithm A Parameters	22
2.	Effects of Variations in Algorithm B Parameters	31
3.	User Level Organization For Algorithm C	35
4.	Formation of The Queue In Algorithm C	38
5.	Flow Diagram of The Scheduler Section of The Simulator	48
6.	Simulator Generated Job Data	50
7.	Simulator Output Data with Early Terminations Allowed, Not Allowed and Background Jobs Permitted	53
8.	Simulator Output Data With Number of Users Variable	56
9.	Simulator Output Data With Arrival Time Average Variable	59

1.0 Introduction

The increasing demands upon computing facilities have brought about the need for more effective usage of present and proposed facilities.

As the number of users and user applications grow, it becomes apparent that a departure from the normal type of design and usage will be necessary, in order to meet expected future demands upon computing facilities. This is true in the case of some facilities even now.

A solution to this problem is the multiprogramming type of system design along with closer coupling between man and the computer. Multiprogramming is defined by Critchlow (6) as, "The time-sharing of a processor by many programs operating sequentially." This means that many object programs are in memory (core or auxiliary), but only one program at a time is actually being executed.

The term multiprocessing is sometimes confused with multiprogramming even though the two are generally regarded as distinct concepts. For the purpose of this paper multiprocessing will mean, the time-sharing of the facilities of a computer by many programs or parts of the same program with the aim of achieving maximum use of each and every facility. Multiprogramming only will be considered in this paper.

The control of the object programs utilizing a multiprogramming system is the job of a supervisory control program, which is generally called the executive routine. The main tasks of the executive routine are: (1) The determining of which object program to execute next and for how long.

(2) The exchanging of object programs between primary and secondary memory.

(3) The control of all I/O operations.

(4) The protection of all portions of memory, belonging to one program, from harm by another program.

The efficient operation of any computer system depends to a large extent on the efficiency and correctness of its executive. This is even more essential when applied to multiprogramming systems, where each and every minute operation of all of the user programs must be scheduled and made to occur in a particular sequence that will maximize the systems effectiveness.

A portion of the executive must be allocated the task of scheduling these operations according to some algorithm, that will give the optimum system performance consistent with the systems goals. When used in this paper scheduling will mean, the determination of the sequence in which job programs will use the available facilities of the computer system.

This paper will investigate some of the methods of analyzing the operation of a scheduling algorithm, point out their capabilities and shortcomings, and propose a general scheme for combining the better of the methods into an organized analysis of an algorithm. This general scheme of analysis will enable a scientific determination of the proper scheduling algorithm for a multiprogramming system.

The investigation will consist of a general examination of each method of analysis and examples of the use of the heuristic and statistical methods on several representative types of scheduling algorithms. The statistical method will utilize a systems simulator based on the Monte-Carlo Method.

2.0 Multiprogramming Background

The terms, time-sharing and multiprogramming, are generally used in such a manner as to imply a common definition. This definition is taken to be the one given earlier for multiprogramming. For the purposes of this paper time-sharing will be used to imply the sharing of a facility of the computer system by many programs, on a time basis, and thereby will become a tool or subset of multiprogramming.

2.1 Motivations for use of Time-Sharing

There exist many motivations for the use of time-sharing in a computer system. Some of which are:

(1) Better man-machine interaction.

(2) More efficient usage of the computer system.

(3) Faster turn around time (the time between formulation of a program and return of the results).

Foremost among these is the desire and need for a much faster man-machine interaction rate. Computational speed has been steadily increasing, while the methods and speeds of interaction between the user and the computer have remained relatively unchanged. The desire to increase the man-machine interaction rate stems from the philosophy that every man-made system is designed to be a "mechanical extension" of man and help him in the performance of his work. Obviously, the closer the coupling between the system and man, the more efficiently and accurately he is able to utilize the capabilities of the system.

The normal type of computer system processes data and solves preformulated problems according to predetermined procedures. During the course of computation all alternatives must be considered and covered or else the program may not operate properly and erroneous results may be obtained.

This is a very exhaustive and lengthy program formulation process, which may consist of many trial runs on the computer before all alternatives are covered and the program is properly formulated. It would be much easier and faster if the user could utilize the capabilities of the computer in the formulation of his program. The formulation phase could then take the form of a guided trial-and-error process in which the computer would turn up flaws in reasoning or reveal unexpected alternative paths in the flow of the program.

By time-sharing the computer between several users of the program formulation type, not only would the rate of man-machine interaction be increased but the utilization of the computer would be increased also, as any "dead-time" (unused computer time), brought about by one user analyzing a computer output to him or formulating his next input to the computer, could be utilized by another user.

Any one of the above mentioned motivations would seem to be justification for a multiprogramming system and certainly a system that could accomplish all three should be very worthwhile indeed.

2.2 System Goals

Multiprogramming, by its definition, covers a vast area in the

computing field. Almost any computer system that time shares any of its equipment or units among many programs may lay claim to being a multiprogramming system of a sort. These various types of multiprogramming are best segmented into groups according to the goals of each system. There exist several fairly well defined goals and many combinations of these with which to segment the types. Some examples of these goals are:

(1) Providing the required service to as many users as possible is one of the basic goals, which defines a type of multiprogramming system that is receiving more and more attention each day. This is as it should be for the computer system is a man made tool and exists for the purpose of helping man solve his problems. Any method of providing better service to man by one of his tools should receive a maximum of attention and effort.

(2) Providing the maximum utilization of each and every unit or facility of the computer system in such a manner as to maximize the efficiency of each unit and that of the system as a whole. The efficiency of the computer system is a very worthy consideration, especially with the vast increase in the usage of computers that is occuring today. Computer systems now have to service more users, who in turn utilize larger and more complex programs than were required a few years back. This necessitates the fullest use of all the facilities of the computer system.

(3) Providing the required service to the control type of users is in actuality a subset of the first example, but is of such importance as to merit its own separate mention. This type of multiprogramming is seeing more and more applications in the military and industrial fields. The problems involved are difficult and will require much effort and study. For example, how does a computer system provide guaranteed response at specific times on a regularly recurring basis to many users? This question needs to be answered in order for this type of multiprogramming to be fully realized.

The determination of goals of a multiprogramming system is of great concern in the formulation of the executive routine and in fact specifies the overall picture of the results that the executive routine must obtain.

2.3 Scheduling

While multiprogramming appears to go a long way toward solving some of the computer usage problems, it contains many problem areas that will require much thought and effort within its own framework. Some of these are outlined by Corbato (4):

(1) Scheduling algorithms.

(2) Switching versus swapping of programs.

(3) Storage allocation.

(4) Memory and input/output protection.

(5) Secondary memory usage schemes.

Each of these are necessary, integral and interdependent sections of a multiprogramming system. The inadequacy of any section can drastically lower the effectiveness of the system, regardless of how perfect the other sections perform. Of prime importance, however, is the scheduling section of the multiprogramming system's executive routine.

2.3.1 System goals as design considerations

Because the scheduler decides which user shall receive what service and when it, of necessity, must clearly reflect the goal or goals of the system. In fact, it may be considered to state the goals in its algorithm. For example, if the scheduler considers only the fact that a user or users are requesting service and honors them in some manner that does not depend on who the user is or where it came from, it then must have as its goal the optimization of service to all users without regard to priorities. This type of system would not be good for control type user programs, nor would it accomplish much toward optimizing the computer system efficiency. Consider though, the system that is concerned with computer system efficiency. Its scheduler would have to consider the past, present, and if possible, future work load of each major unit of the computer system. It would have to determine the best mix of jobs or job segments that would keep each unit as busy as possible.

It is apparent then, that a very necessary prelude to the design of a scheduling section for a multiprogramming system's executive routine, is a careful and precise determination of the goals of the system. When this is done the task of the scheduler, in most cases, will be outlined

and only the problem of accomplishing it remains. As in the case of much of scientific design, the answer to the question, "What exactly is the problem," is not only the first, but sometimes the most difficult task encountered.

2.3.2 Overhead Considerations

The justification of the overhead created by any section of the executive routine is a very critical factor. It is even of a more critical nature for the scheduler than for other sections, as the scheduler is used more often than the other sections of the executive routine. It serves no purpose to design a scheduler that considers every aspect of the data concerning the user programs and the units of the system, calculates the effect on the realization of the goals of the system and goes through a complicated process to maximize the realization, only to the end of taking up more computer time for the process than was saved (either for the user or the system). This is a clear case of too much overhead and cannot be tolerated. The ambitions of the scheduler should be lowered, in this case, and a level reached, so that the savings more than exceed the overhead.

2.3.3 Scheduler Input Considerations

Another problem area in the design of a scheduler is the input information required for the operation of the scheduler. This must be kept within reasonable bounds as some information requirements may be either impossible or impractical. For instance it would be desirable to have the expected running time of each user program requesting service, but this fact is not known unless the program has been run before with the

same input parameters, a rare occurrence indeed.

2.3.4 Cost-effectiveness Considerations

In all aspects of the design of a scheduler a cost-effectiveness study must be made and the question, "What does this feature buy and cost the system and/or user?", must be answered. The cost-effectiveness criterion as applied to this answer will determine whether or not the feature should be adopted. The cost-effectiveness criterion is another item that is very dependent upon the goals of the system. The system that is oriented towards the control type user, for example, will have a cost-effectiveness criterion that demands the adoption of any feature enhancing the system's ability to guarantee the user service at a specific sampling rate.

2.3.5 Summary of General Scheduler Design Considerations

A brief summary of the generalized scheduler design problem is:

(1) There are many various types of schedulers and each is specified by the particular goals of the overall system it serves. Therefore, specify the goals of the system concisely enough and the type of scheduler and its task is specified.

(2) The overhead of a scheduler must not be allowed to get out of hand and the complexity of the scheduler decreased if it does.

(3) Input requirements to the scheduler should be carefully examined for practicality and necessity.

(4) A cost-effectiveness study should be made of every feature proposed for the scheduler before its acceptance. The cost-effectiveness criterion must reflect the goals of the system.

3.0 Investigation

The practical aim in investigating a scheduling system by any form of analytical method is usually to analyize the actions of the system and try to improve it by appropriate changes. For example, the rate of arrival of user requests may be so high that large queues develop, resulting in long delays in service for the users, or the rate of arrival requests may be so low that the facilities of the computer system are idle for a large portion of the time. A change in the scheduling algorithm may be indicated in either case. Or it may be that a change in the scheduling algorithm is being contemplated and predictions of the amount of change in service capabilities are needed. In any event frequent analysis of the operation of a scheduler is needed to insure its continuing operation at optimum level.

3.1 Methods of Investigation

There are several methods of investigating the operations of a scheduler available to the systems designer, each of which has its own particular areas of maximum effectiveness. These methods may be divided into the following categories:

(1) Heuristic approach.

(2) Queuing theory approach.

(3) Statistical simulation approach.

(4) Actual system implementation approach.

3.1.1 Heuristic Approach

The heuristic approach generally consists of a mathematical and logical examination of the operation of a scheduler without the necessity

of gathering large amounts of data for analysis. As a result, it is easier to accomplish, but is not as rigorous as other methods, nor does it give as complete picture of the operations. It may, however, turn up many problem areas and indicate possible directions for further investigation.

3.1.2 Queuing Theory Approach

The queuing theory approach is an analytical method that furnishes theoretical predictions of the operation of the scheduler. These predictions are based on a stochastic model of the scheduler composed of mathematical formulas derived from the probabilistic nature of the scheduler environment. This model requires inputs of a statistical nature and therefore necessitates the investigation of fairly large amounts of data concerning the system. For example, the arrival rate pattern of user requests, the service rate pattern, the rules governing service and the rules governing the queue discipline. The basic weaknesses of the queuing theory approach lie in the assumptions that sometimes have to be made concerning the exact probabilistic nature of the environment and its inability to handle scheduling algorithms that specify much interdependence between the various patterns mentioned above.

3.1.3 Simulation Approach

The simulation approach is more of a representation of the behavior of the scheduler than a mathematical analysis, as is the case with the heuristic and queuing theory approaches. The simulation approach does, however, utilize the theory of probability and statistics in simulating the environment of the scheduler.

The simulator is an ideal method to use in place of queuing theory when there exists much interdependence between the parameters of the scheduler environment. The type of detailed results obtained from simulation will give a direct qualitative impression of what the behavior of the system should look like.

3.1.4 System Implementation Approach

The actual implementation approach consists of placing the scheduling section in its entirety within the actual multiprogramming executive routine and testing its operation within the complete system. This method is not to be recommended as a first method of investigation of a scheduler with new changes as it is extremely costly in time and effort while also causing the disruption of the normal operation of the system during the testing phase.

3.2 Proposed Composite Investigation Procedure

Each of the previously mentioned methods has its own place in the overall procedure of the investigation of a scheduler and contributes its particular type of information to the total collection of intelligence gleaned from the investigation.

An outline of such an overall investigation procedure is as follows:

(1) Preliminary survey.

(2) Collection of statistical data.

(3) Detailed study of the system and effects of any changes to the system.

(4) Small scale trial of newly proposed system.

(5) Full scale implementation of proposed system.

3.2.1 Preliminary Survey

A preliminary survey of a scheduling algorithm would entail a brief, but careful study of the algorithm and its basic philosophy. This study would be of the heuristic type and should reveal the limitations of the algorithm and point to avenues of further and more detailed investigation, if not to possible modifications themselves. If modifications are indicated, they should be roughly assessed and if of merit passed on for a more comprehensive investigation.

3.2.2 Collection of Statistical Data

The next step involves the collection of statistical data for use with a queuing theory analysis of the scheduler or a simulation model of the scheduler.

The statistical data gathered is generally segmented as follows:

(1) The arrival pattern data gives the statistical pattern of the arrival of user service requests and is generally in the form of a distribution function, mean arrival rate and deviation from this rate.

(2) The service mechanism data concerns itself with the number of users that can be serviced during some period of time and the statistical pattern of the length of service time, which consists of a distribution function and its necessary parameters.

The arrival pattern data may be separated into requests for various types of service and the service data separated in an analogous manner. This data is best obtained by proper use of statistical sampling methods.

3.2.3 Detailed System Study

To conduct a detailed study of the scheduler and its operations normally requires the use of a simulation model. The queuing theory approach will in general be much harder to implement and not give as correct data due to the complexity and interdependence of parameters found in most scheduling algorithms.

The correct simulation model will give data that follows closely the operations of the actual system. Thereby allowing an investigation of the present algorithm to be made under easily varied conditions and suggesting modifications that may be checked out in the same manner.

3.2.4 Small Scale Trial

The modifications that have passed the investigation of the simulation model should then be incorporated in the executive routine on a small scale and given trial runs to more accurately judge their compatability with the rest of the system and their effectivenesses. This procedure may be skipped if the modifications are slight and it is readily apparent that they will be easily implemented on a full scale basis. However, this would be an exception and time and effort will usually be saved by a small-scale trial run.

3.2.5 Full Scale Implementation

When the modifications have been checked out by a small-scale trial run or two, they should be incorporated fully into the executive routine and utilized as much as possible for many hours with close observation and collection of data in order to judge their effectivenesses.

The close adherence to these procedures in the order outlined, will allow the adoption of a new scheduler or modification of an old one, with full assurance that it will not disrupt the system and will bring about a closer realization of the goals of the system.

3.3 Preliminary Survey by Heuristic Analysis

As stated earlier a preliminary survey of a scheduling algorithm is a necessary prelude to further investigation of a new algorithm or changes to an old one. This survey may only point the way for further more precise investigation or may show the need for changes by itself. In most cases, however, it will be advisable to continue the investigation with more precise methods.

For the purpose of this paper a preliminary survey of several different types of scheduling algorithms will be presented. These different types of algorithms will be representative of algorithms for use in multiprogramming systems each having goals of the types described in section 2.2.

No attempt will be made to prove one algorithm better than another as each will have its own distinct goals and will be designed to optimize them. It will, instead, be the aim of this section to point out areas of investigation and changes that would appear to lead to a better realization of the systems goals or a broadening of the goals to include some worthwhile feature.

The algorithms presented will generally concern themselves more with the user's interests and needs rather than system efficiency as this appears to be of the major concern at the present time.

Three algorithms will be presented, one of the type developed by S.D.C., one proposed by Corbato (4), and one conceived by the author.

3.3.1 Algorithm A

One of the first categories of scheduling algorithms to examine should be one which is intended for use in a multiprogramming system with optimization of user service as its goal. An algorithm of the type used by System Development Corporation within their "Time-Sharing System" at Santa Monica, California is an excellent example of a member of this category. This algorithm will be referred to as Algorithm A hereafter.

3.3.1.1 Operation of Algorithm A

The execution of many user programs for a quantum of time (q) in a round-robin type of operation, is the basic principle of this scheduling algorithm. A round-robin queue is set up for each response cycle with the users in the queue being those that requested service during the last response cycle time. The quantum or time of operation of each program during the cycle is determined by the ratio of the response cycle time to the number of users in the queue. The cycle time (t_r) is chosen to be approximately the average human response time in order to achieve the effect of simultaneous use of the computer by many users.

The maximum number of users (Nmax) is determined, in part, by the minimum allowable quantum time (qmin). This time is bounded by the amount of time required for the average user program to produce a response (I/O operation).

If a program does not use up its alloted quantum time in computation, but is terminated by a I/O or finished interrupt, the quantum is recalculated for the remaining users in the response cycle. The redistributing of the remaining time in the response cycle not only increases the efficiency of the system, but also allows the larger, slower user programs more execution time. If at any time during the cycle no users are in the queue, the system goes into an "idle" loop and waits until a user requests service (allowable during every interrupt of the quantum clock).

Priority of program service during a given interval of time is determined by a program's size and its use of low speed I/O equipment. The user programs occupy different banks of core memory according to their priorities. The priority levels, in the order of service, are: (1) programs of less than 16K and not using low speed I/O, (2) programs of less than 16K and using low speed I/O, or programs between 16K and 32K and not using low speed I/O, (3) programs in excess of 32K.

This scheduling algorithm is well suited and very effective for system use of the type that consists, primarily, of tasks that allow a high percentage of the user's system time to be taken up by the slow human thought and reaction process. Examples of such tasks are: (1) on-line debugging, (2) on-line calculation, (3) on-line program building and (4) war games.

The main concern of a scheduling algorithm, when dealing with these types of tasks, is to furnish the users with reasonable response to service requests and reasonable reaction times. The efficiency of the

computer system is considered, but not allowed to become of primary importance.

3.3.1.2 Effects of Varying Basic Parameters of Algorithm A

There are several significant parameters that may be examined during a preliminary survey of this algorithm. They are: (1) maximum number of users, (2) response times, (3) quantum times and (4) efficiency.

By the use of variations of one basic formula much information can be obtained concerning these parameters. The formula is of the "Worst-case" type in that it gives a pessimistic view of the operations based on values of the formula parameters that represent the worst conditions possible. This type of analysis is very helpful, as it shows the lower bound of service capabilities and insures good operation if expectations do not exceed this lower bound by significant amounts.

For determining the maximum number of users allowable with the minimum quantum, with the percentage of overhead (n) and the "Worst-case" value of the time it takes to transfer a program from secondary to primary memory or vice versa (t_s) either known or approximated, the following formulas apply.

Equation 1

$$Nmax = \frac{t_r(1-n)}{2t_s + qmin}$$

Equation 2

$$Nmax = \frac{t_r\ (1-n)}{2t_s}$$

These formulas are derived by analyzing the basic cycle of operations concerning one user during a response cycle.

First it is determined, which user is to be executed for the particular quantum of time and his program is brought in from secondary memory, after the last operating program is transferred to secondary memory. This takes an amount of time equal to $2t_s$. The program is then operated for one quantum (q). Of course, deciding which program to bring in and other executive functions take up a certain percentage of the cycle time (t_T) and have to be accounted for under the heading of overhead (n).

Clearly, then, the denominator of the Equation 1 represents the time necessary to execute one user during the response cycle while the numerator represents the effective operation time available during the response cycle. It must be emphasized that Nmax is the maximum number of active users in the system i.e., the number in the queue. There may be many more actually using the system as long as not more than Nmax request service during the same response cycle.

In order to investigate ways of increasing Nmax, the effects of changes in the various parameters of Equation 1 on Nmax must be analyzed. It will make the investigative task simpler, if the parameters are broken up into groups according to whether they are easily changed by a software approach or require hardware modifications to induce significant changes.

The swap time (t_s) is the only parameter that would most likely require hardware modifications to measurably change it. For instance, the swap time may be decreased and Nmax increased by increasing the transfer rate between primary and secondary memory, allowing two or more programs in memory, or both. Increasing the transfer rate is obviously a

hardware change and is usually a major one if substantial increase in rate is to be accomplished. Allowing two programs in memory, at first glance, does not seem to require hardware modifications, but on further analysis it will be seen to require memory protection to prevent interaction between the two programs and dynamic memory relocatability, in order to achieve the desired results.

The main thought concerning hardware changes, whether they are modifications to existing hardware or design features for proposed equipment, is to be sure that the increase in capability is substantial enough to warrant the cost of the change.

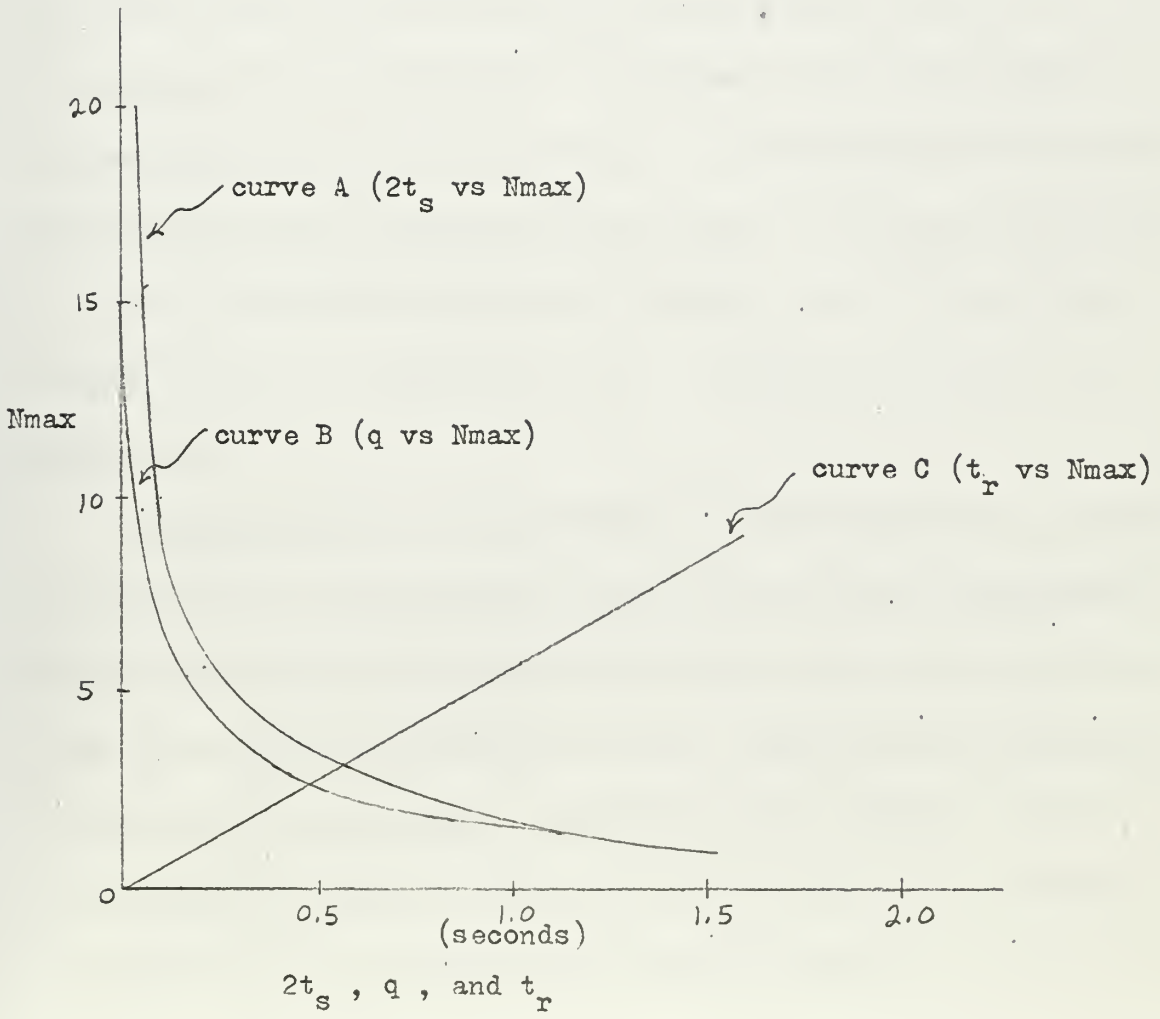
The technique of showing graphically the capability increase versus change in parameter values is an invaluable tool to use in determining whether or not further study is warranted. This technique is illustrated in Figure 1 where curve A represents Nmax versus primary to secondary memory transmission rate and curves B and C show the relationships between qmin and t_r, respectively, and Nmax. The latter two variables, qmin and t_r, are variable by software means and depend more on user wants and needs for their values than on the computer system limitations. This means that the cost of their change will be in terms of changes in some user service parameters rather than monetary values or man-hours.

The estimation of the effect of allowing two or more programs in memory at one time actually involves the modification of Equation 1. If memory sharing is allowed and qmin is substantially less than $2t_s$, then the limiting factor in the denominator of Equation 1 becomes $2t_s$ and

Figure 1

Effects of Variations in
Algorithm A Parameters

$$Nmax = t_r(1-n)/(2t_s+q)$$



Equation 1 shifts to the form of Equation 2. The minimum quantum (qmin) effectively vanishes because concurrency is now possible between the transferring of one program and the execution of another. If qmin were much larger than $2t_s$, then $2t_s$ would likewise effectively vanish, but this will hardly ever be the case.

Whether or not $2t_s$ is substantially larger than qmin depends upon the distribution of user requests and can only be roughly estimated without such data. From observations of the "Time-Sharing System" at S.D.C., $2t_s$ does not appear to be much larger than qmin and therefore the effect of memory sharing on Nmax does not seem to warrant its cost. Of course, this applies to this particular system and its users and the outcome of the analysis might be entirely different for another system and its users. The important thing is that the method of analysis will remain the same.

For a complete analysis of the algorithm, other pertinent characteristics, such as, response time and computational efficiency, must be investigated.

Response time, for this algorithm, may be considered t_r, for worst case analysis. In other words, a user requesting service immediately after a new response cycle had started would have to wait a maximum of time (t_r) until he was accepted for service, while one which requested service at the end of a cycle would have to wait a minimum of time. Another form of Equation 1 may be used (Equation 3) for this analysis and graphical results illustrated by curve C of Figure 1.

Equation 3

$$t_r = N(2t_s+q)/(1-n)$$

As may be seen by an examination of the equations and the curves of Figure 1, N and t_r are linearly dependent (if $2t_s$ and q remain constant) and therefore each is very sensitive to any change in the other. As each increases without bound, with any increase in the other, some upper limit must be set for one of them in order to contain the investigation in a workable area. This limit is more easily placed on t_r. The response time should not be allowed to exceed a value that would appreciably degredate the service needed by the type of user for whom the system has been designed. In the case of this particular algorithm, the maximum t_r is an approximation of the human response interval. This then confines the problem to a smaller working area and families of curves A, B, and C may be drawn within this area for closer analysis.

It is clearly shown by the graph that if the major concern is to maximize the number of users, then t_r must be set at its maximum value. On the other hand, if servicing a user with particular needs as to response time is of primary interest then N must be restricted to a value that will allow the needed t_r. In most cases a combination of these goals is present and a compromise is reached between Nmax and $t_r$max with the other parameters being varied in order to ease the conflicts. This is the case with the particular algorithm in question as t_r is restricted to a maximum value, but allowed to vary up to this maximum as the number of users dictate.

The quantum time is allowed to vary, but has a minimum value that is normally arrived at by a compromise between computational efficiency

and maximizing the number of users. Clearly as q is decreased the number of users allowed (N) goes up, but the more often a program is transferred in and out of memory the more overhead is increased, swap time per program is increased and computational efficiency is decreased. The term computational efficiency is taken to be a measure of the ratio of the time a user program spends in the computational phase to the time it spends in preparing for the computational phase. The latter time does not include time spent waiting for a normal user turn. The following equation is one method of expressing computational efficiency (C.E.).

Equation 4

$$C.E. = \frac{q(1-n_q)}{2t_s}$$

The parameter n_q represents the portion of overhead that may be considered taken up by each user everytime his program runs for a quantum of time. Because of the fact that t_s is not easily variable computational efficiency depends to a great extent on quantum time and is maximum when q is maximum.

It is important to bear in mind that rarely will a multiprogramming system achieve a computational efficiency greater than 50% and most do not even reach this. Computational efficiency, therefore, should be maximized, but not at the expense of user service and should not be used as a major comparison parameter between systems.

The minimum quantum time is determined not only by consideration of computational efficiency (as illustrated in Equation 4) and the number of users allowed, but also by a consideration of obtaining the maximum

use out of every assigned quantum of time. It does not help to assign a user a larger quantum of time in order to gain better efficiency only to have him terminate his quantum by an I/O operation after only a few milliseconds. Nor does it help to assign a user a smaller quantum and have him need only a little more to come to a logical termination. These situations can not be avoided for every user, but by a careful study of user distributions as far as program run time between I/O operations is concerned and anslysis of the effects of varying q, a suitable compromise between efficiency, and user service may be reached.

3.3.1.3 Summary of Analysis of Algorithm A

The following comprise a brief summary of the heuristic analysis of Algorithm A.

(1) Either an excellent intuitive feel for the expected distribution of such thing as, user arrival rates, time between I/O operations and response time requirements or else actual distribution data should be available for a preliminary analysis.

(2) The user service oriented scheduler must set a maximum limit on t_r that is based primarily on user response requirements.

(3) A minimum limit on quantum time must be set consistent with proper utilization of the quantum (based on user time between I/O's distribution), number of users allowed in each response cycle and computational efficiency. The emphasis should be placed on the first two conditions.

(4) Unless much faster transmission rates between primary and secondary memory are available, memory sharing does not appear to increase the

capabilities of the system enough to warrant its implementation.

(5) A simulation study of the effects of not allowing early I/O terminations to cause a recalculation of q would be helpful in determining if the overhead involved were prohibitive.

(6) A simulation study of the effects on computational efficiency and user service due to the modification of the algorithm to allow background jobs to fill in when the system has no other users would also be advisable.

3.3.2 Algorithm B

The next type of algorithm presented for analysis is one which also has as its goal optimum user service, but does give consideration to computational efficiency.

This algorithm (3) was developed at Massachusetts Institute of Technology and was one of the forerunners of the field. For the purpose of this paper the algorithm will be designated Algorithm B.

Algorithm B is what is considered a multilevel scheduling algorithm, as the scheduling process sets up multiple level queues of programs waiting to be serviced.

3.3.2.1 Operation of Algorithm B

The multilevel scheduling algorithm is implemented in the following manner:

(1) Each user program is assigned to the L_o th level priority queue, as it enters the system.

Equation 5

$$L_o = \lceil \text{Log}_2(\lceil w_p/w_q \rceil + 1) \rceil$$

w_p = number of words in user program

w_q = number of words that can be transferred in and out of primary memory during one quantum (q) of time

$\lfloor \overline{A} \rfloor$ = Integral part of A

(2) The program at the head of the lowest level queue (L) is executed for up to 2^L quanta of time and if it is not completed is placed at the end of the $L+1_{th}$ level queue to be executed for up to 2^{L+1} quanta of time later on After execution of all of the L_{th} level programs for 2^L quanta, programs in the $L+1_{th}$ level are executed. If a program at the $L+1_{th}$ level is being executed and the L_{th} level becomes occupied by an active program, the presently executing program is placed at the head of the $L+1_{th}$ level queue and the L_{th} level program is brought in and executed. If there are no levels active, background jobs are brought in and executed until a level becomes active, thereby increasing computational efficiency and lowering idle time.

3.3.2.2 Effects of Varying Basic Parameters of Algorithm B

From Equation 5 it can be seen that program size is an essential input into Algorithm B and therefore requires that user programs be pre-compiled.

Further study of the equation shows that L_o can never be less than zero due to the fact that (w_p/w_q+1) has one as its lower limit. With (w_p/w_q) q representing the swap time $(2t_s)$ of the program, then L_o is always at least slightly greater than $\text{Log}_2 (2t_s/q)$. This coupled with the practice of executing the program for $2^{Lo}(q)$ or, as $(2^{\text{Log}_2(2t_s/q)})\ (q) = 2t_s$, for greater than the swap time of the program,

insures a computational efficiency of at least 50%.

A computational efficiency of at least 50% arises from this analysis only if overhead time is neglected. As overhead can never be neglected, an overhead factor (n) must be included in any equation concerning efficiency.

Equation 6 may be taken to represent computational efficiency with overhead included for programs operating from the L_{th} level queue.

Equation 6

$$C.E. = \frac{Nq^{2L}(1-n)}{N\,(\overline{\angle w_p/w_q \angle})^q} = \frac{(\overline{\angle w_p/w_q \angle} + 1)\,(1-n)}{(\overline{\angle w_p/w_q \angle})}$$

Where N = number of users at Level L

n = fraction of overhead involved for each user/quantum

As may be seen Equation 6 reduces to the point where computational efficiency appears to be independent of q and N. This is not strictly true as n is a proportional parameter and is based on a particular q. This then imposes a limit on q that requires it to remain large enough to keep the overhead to a low proportion of q and computational efficiency close to 50%.

Another equation that is helpful in analyzing Algorithm B is one that describes the calculation of the worst-case response time (t_r) for each user when all N users are on one level.

Equation 7

$$t_r = 2Nq\,(\overline{\angle w_p/w_q \angle} + 1) - Nqn$$

$$= Nq \quad 2\,(\overline{\angle w_p/w_q \angle} + 1) - n$$

Equation 7 is arrived at by considering that q ($\lfloor \overline{w_p/w_q} \rfloor$ +1) is both the swap time and the execution time for each program and that qn is the overhead time for each program. With N programs at the particular level, N times the sum of the swap time, execution time and the overhead time will determine the length of time any user has to wait between executions of his program.

Equation 7 shows that response time is directly proportional to both the number of users and quantum time. Quantum time has already been shown to have a lower bound fixed by efficiency considerations and now response time (or Nmax) considerations create an upper bound. It is very apparent that a compromise must be reached between the number of users allowed at each level (Nmax) and the required response time (t_r), just as in the case of Algorithm A.

A graphical picture of the various relationships expressed in Equation 7 is shown in Figure 2.

3.3.2.3 Summary of Analysis of Algorithm B

The following statements may be used to summarize the preliminary survey of Algorithm B.

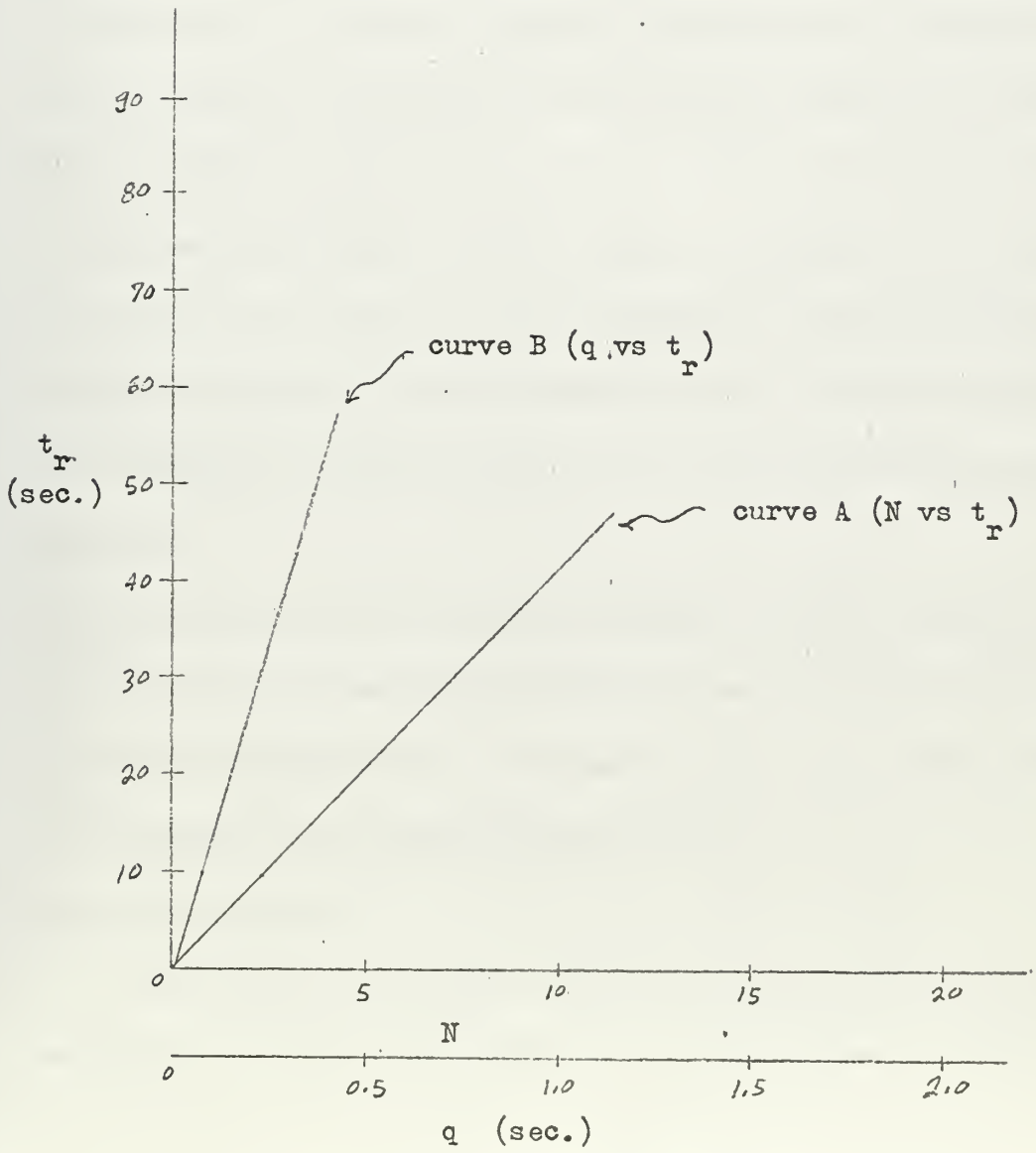
(1) The quantum time is chosen in approximately the same fashion as for Algorithm A with the lower bound set by efficiency requirements and the upper bound set by maximum allowable user considerations.

(2) Response time and maximum number of users must be determined by a compromise between the optimization of each.

FIGURE 2

Effects of Variations in
Algorithm B parameters

$$t_r = Nq(2(w_p/w_q + 1) + n)$$



(3) All of the equations concern one level only and consider all programs in the system to be on this level for worst-case analysis. To accomplish a proper analysis considering the users spread about on multilevels is a complex undertaking and is best left to a simulation study.

(4) It is possible for users with large programs to get pushed up to a high level and never get to run at all if there exist many small users with relatively short times between I/O operations. This could be corrected by insuring that every user in the system was allowed to run at least once during some cyclic time regardless of his level position.

(5) The large user program, which has fast reaction time (time between I/O's) is unduly restricted by its size as it can never go to a lower level than the one it entered upon and may use only a small portion of its alloted execution time, while still having to wait the full t_T specified by its high level before its next execution turn. There should exist some compromise between raising the level for larger and longer running programs and lowering it for programs that consistantly terminate their execution time early.

(6) The background user approach represents a goodly increase in efficiency particularly if the type of foreground users are those who require periods of thought process between I/O operations and thereby leave a fair amount of idle time for use by the background user.

3.3.3 Algorithm C

There exists a need for a scheduling algorithm that can handle efficiently a variety of user types such as: (1) production job user,

(2) man-machine communications user and (3) sample-data control user.

A production job type user is typified by his ability to encounter fairly long delays in obtaining service without harmful effects.

The man-machine communications user needs to have a response time somewhere close to the human reaction time, but is not hurt badly if he misses a turn or two occasionally.

Sample data control users are very critical concerning response time. These users must be assured of a certain fixed (for each different user) interval between its samples or execution turns. This puts an extreme strain on the scheduler and requires a very stringent priority queuing scheme.

The task of trying to serve all three of these types of users necessitates the design of a scheduler that tries to meet all three of the goals outlined in Section 2.2. Of course, any attempt to maximize three different forms of service as one is bound to result in a system that compromises and does not fully maximize each individual type of service. This is to be expected and resultant service degredation to any one type of user must be accepted as the price required for a generalized rather than specialized system.

The algorithm proposed for this generalized type of system, Algorithm C, is a three-level scheduling algorithm, where the various levels are defined by the type of users assigned to each one. The levels and their associated programs have inherent priorities and reside in

FIGURE 3

User Level Organization
For Algorithm C

level and priority	user type	waiting storage
I	sample-data control	core
II	man-machine communications (I/O stations)	drum
III	production jobs (background)	disc

different types of storage when awaiting service. Figure 3 illustrates the level organization.

3.3.3.1 Operation of Algorithm C

The operation of the scheduling algorithm is as follows:

(1) At the beginning of a response cycle, the number of user requests for each level is determined (only one user each from levels I and III are allowed per cycle).

(2) The sample rate and execution time of the level I user will have to be known and the quantum time (q_1) will be equal to or slightly greater than his execution time, just as long as the reciprocal of the sample rate (t_s) is an integral multiple of the quantum time. This is in order to be able to insure the level I user of service at exactly his specified sample rate.

(3) The total execution time of the level I user is subtracted from the cycle time (which is also an integral multiple of calculated q_1) and the remaining time is allocated to level II and III users on the basis of a q_2 (integral multiple of q_1) basis. The quantity, q_2, is not allowed to exceed t_s.

(4) Only one q_2 is given to the level III user unless there are not enough level II users to fully utilize the allotted time, in which case, the level III user is given the remaining time.

(5) Early terminations of q_2 by level II users I/O operations are not allowed to change the structure of the queue nor cause recalculation of q_1 as this would result in the inability to insure the level I user of a specified sampling rate.

(6) If a level I user is requesting service, a new cycle execution may not begin except at the time specified, by the level I users sample rate, for the next level I user sample.

Figure 4 shows the manner in which the queue is formed for Algorithm C.

(7) The queue should both begin and end with a level I user in order that overhead time involved in setting up the queue, etc., will occur between level I samples and therefore not unduly degredate sampling rates.

As the number of users from levels I and III are restricted to one each per response cycle, it is fairly easy to express, in equation form, a relationship concerning the number of level II users. This equation is specified by the following parameters:

(1) t_c - time of one response cycle.

(2) n - percentage of overhead per cycle.

(3) N_2 - number of level II users.

(4) N_1 - pseudo number of level I users or sample rate divided by cycle time $(1/t_s t_c)$.

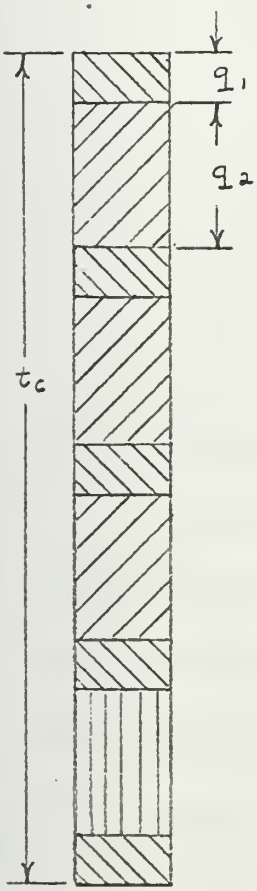
(5) t_2 - time to transfer an average level II user to or from core memory.

(6) t_3 - time to transfer an average level III user to or from core memory.

(7) q_2 - quantum time for level II users (Mq_1, where M is an integer).

FIGURE 4

Formation of the Queue
in Algorithm C



Legend:


— level I user


— level II users


— level III user

order and length of
execution of user
programs by Algorithm C

The number of level II users allowed (N_2) is equal to the amount of effective cycle time ($t_c(1-n)$), minus the time to transfer user programs during the cycle ($N_2 2t_2 - 2t_3$, $N_3=1$), divided by the level II quantum time (Mq_1), minus the number of level III users (1) and the pseudo number of level I users prorated to a number utilizing a level II quantum time (N_1/M). From this description of the number of level II users comes:

Equation 8

$$N_2 = \frac{t_c(1-n) - N_2 2t_2 - 2t_3}{Mq_1} - (1+N_1/M)$$

or

Equation 9

$$N_2 = \frac{t_c\,(1-n) - 2t_3 - q_1(M+N_1)}{Mq_1 + 2t_2}$$

3.3.3.2 Effects of Varying Basic Parameters of Algorithm C

As might be expected from the description of the operation of the three level scheduling algorithm, the specifications concerning the level I user's needs dominate the algorithm. This is reflected in Equation 9 also. Note that the factor q_1M is equal to the time between level I user samples (t_s), the term N_1 is equal to $1/t_s t_c$ and q_1 is in reality the execution time of the level I user. This leaves only the transfer times (t_2 and t_3) and the cycle time that may be varied to maximize N_2. The transfer times are essentially hardware limited variables and therefore are costly to change. Cycle time may be increased to allow more level II users, but will cause a corresponding increase in level II users response time as t_c

is the response time for all but level I users. In short the critical factors in this algorithm are controlled by the level I user and variations allowed are few. This will, in general, be the case for any algorithm that concerns itself with control type users, as there is very little flexibility in their service needs.

There are several limiting factors inherent in this algorithm and the services that it can provide. First of all consider the limitation on the maximum sample rate allowed the level I user. One of the factors connected with the sample rate limitation is the average time necessary to transfer a level II user to core and let him accomplish some worth while computation.

With a fairly fast computer having two high speed I/O channels and capable of having several programs in memory at one time, this time may be controlled to where it is not the limiting factor. With the above mentioned capabilities the transfer times are, on the average, effectively "swallowed" by the quantum of computational time due to the allowance of concurrency of compute, input and output operations.

The main limiting factor then comes to be the overhead time between response cycles as, with the level I user having a quantum of execution time at the beginning and the end of the cycle, the overhead time effectively represents the sample time. It is estimated that this value could be kept in the neighborhood of five milliseconds for computers with average instruction times of around three microseconds. This would allow 200 samples per second or effective sampling of 100 cycle per second

functions. A goodly number of control problems may be specified within this frequency range and investigated under multiprogramming.

3.3.3.3 Limitations of Algorithm C

The major drawback of the algorithm is that, as shown above, two high speed I/O channels, extensive and flexible memory protection and large core memory is required in order to achieve decent service capabilities for the level I users. These requirements are extremely expensive and are beyond the reach of many installations. One further drawback is the reduction of the level II and III users quantum time in order to gain better sample rates for the level I user.

3.3.3.4 Proposed Changes to Algorithm C

A possible solution to the problem would involve forming the cycle for the level II and level III users alone and then allowing the level I user to interrupt, when ever and for as long as he needs, via a separate quantum clock. As the level I user would normally not need a very long quantum, this would not increase the cycle time beyond reason and would allow the level II and III users to stay in core long enough to achieve meaningful computation. In addition the extra high speed I/O channel would not be needed, as with a larger q_2 the transfer times could be accomplished, for the most part, concurrently with computation and would not degredate level I user service in any case. A further advantage is that computational efficiency would go up due to the overhead involved in transfers, etc. becoming a smaller percentage of q_2.

3.3.3.5 Summary of Analysis of Algorithm C

It should be apparent now that the problems involved in including capability to service one-line control type users in a multiprogramming system are many and no easy solution exists that will provide good service for all types of users. Probably the easiest solution to implement is to provide two separate algorithms (one for all three levels of users and one for level II and III users only) for the system that may be switched in or out depending whether or not control type users are scheduled for the particular time concerned or not. This though requires strict scheduling of blocks of time for different types of users and should be considered an interim measure only.

The algorithms involved in the on-line control class are sufficiently complex to be very hard to analyze by simple methods and therefore lend themselves readily to analysis by simulation methods. The main concern in the simulation analysis should be determining what values for the parameters of the algorithm in question give the best level I user service (with respect to sample rate) and, at the same time, the least degredation to level II and III user service. The investigation should be concerned with effects on computational efficiency also.

There exists a great need for work in this area of multiprogramming as universities, research laboratories and even industry would benefit greatly by being able to solve all of their computing needs (control needs included) with one computer system and yet allow each user to feel that he has the system at his disposal at any time.

3.4 Detailed System Study by Simulation

As stated earlier, the simulation approach to the analysis of a particular scheduling algorithm is characterized by its detailed picture of the operations of the scheduler and its ability to provide a method that allows the analysis of an algorithm with a high degree interdependence between its parameters.

The algorithms mentioned in section 3.3 all contain a certain degree of interdependence among their parameters and, while a heuristic approach may bring forth much valuable information, a detailed study of the operations of the algorithms should be conducted by simulation techniques before decisions concerning changes or acceptances are made.

3.4.1 Features of The Simulator

The simulator used for the purpose of this paper is very flexible and can be used to examine any part of the system's operation that may be desired.

3.4.1.1 Input Data Features

The input data consists of such things as:

(1) User distributions as pertain to job arrival times and various service rates.

(2) Job types.

(3) Number of user channels.

(4) Type of scheduler to be used.

(5) Parameter of algorithm to be varied.

(6) Type of output desired.

(7) Length of run (time).

An example of the input data to the simulator may be seen at the end of the program contained in Appendix II.

The part of the input data concerning the user distributions and job types are converted by the simulator to job specifications for each user station. The specifications contain the following information:

(1) Job number-chronological number of the job generated.

(2) Clock time-the time of generation of the job.

(3) Station number-the number of the user station for which the job was generated.

(4) Job type-the type of job generated (specifies the particular set of means used to generate the job).

(5) Arrival time-the time that the job will request service.

(6) Load time-the amount of time needed to load the job.

(7) Active time-the amount of time the job should spend actually computing.

(8) I/O time-the amount of time the job will spend in an I/O phase.

(9) Repeats-the number of times the active and I/O phases will be repeated.

(10) Size-the number of memory cells required to contain the job.

An example of the generated job data is contained in Figure 6.

3.4.1.2 Output Data Features

The output data may be arranged in convenient tabular form or in graphical form depending on which is desired. This capability allows the investigator wide latitude in choosing the data format that will be easier

to analyze for each output parameter.

It will be possible to form comparisons between various schedulers using the same user distributions and determine which one best suits the needs of the system and its expected users by use of the simulator. The simulator also allows the study of the effect of varying any algorithm parameter and can present tabulated or graphical output data for ease of comparison between various values of the parameter in question.

For analysis of scheduler operations, the simulator output data contains:

(1) Variable parameter-Values of the parameter that is varied during the run.

(2) Cycle count-The number of response cycles that occurred.

(3) Average cycle time-The average amount of time taken by a response cycle.

(4) Average number in queue-The average number of active users during a response cycle.

(5) Average quantum-The average amount of time taken by one quantum.

(6) Average overhead-The average amount of time taken up by scheduler overhead during one cycle.

(7) Computational efficiency-The average amount of time spent actually computing per response cycle, divided by the average cycle time.

(8) Average exchange overhead-The average amount of exchanger overhead per response cycle.

(9) Average exchange time-The average amount of time taken up by loading and exchanging per response cycle (does not include overhead).

(10) Average overload-The average number of jobs that were unable to obtain service, due to the queue being full, per cycle.

(11) Maximum quantum-The maximum amount of time taken by one quantum.

(12) Maximum cycle time-The maximum amount of time taken by one response cycle.

(13) Maximum number in queue-The maximum number of active users during one response cycle.

(14) Maximum overhead-The maximum amount of scheduler overhead during one response cycle.

(15) Maximum overload-The maximum number of jobs unable to obtain service during one response cycle.

(16) Maximum number of stations-The maximum number of user stations allowed.

(17) Requests serviced-The number of user requests for service honored.

Figure 7 contains an example of the output format of the simulator.

As may be seen the simulator output easily furnishes enough information for the proper analysis of the operation of a scheduling algorithm.

For further and more detailed information concerning the operation of the simulator consult Appendix I.

3.4.2 Example of Simulator Operation

In order to demonstrate the capabilities and operation of the simulator, an analysis of the scheduling algorithm (Algorithm A) used by System

Development Corporation in their "Time-Sharing System" at Santa Monica, California will be conducted using the simulator. A flow diagram of the scheduler section is contained in Figure 5.

3.4.2.1 Input Data

The input data is based on observations of the operations of the "Time-Sharing System" and while the data is not the result of a thorough statistical study of the system, it will be sufficient for the purpose of this example.

3.4.2.2 Variations of the Algorithm To Be Considered

The preliminary analysis of Algorithm A in section 3.3.1 outlined several variations of the algorithm that would be worth while investigating by simulation techniques. These and others will be analyzed by the use of the simulator and it will be determined if the variations would further the achievement of the system's goals or not.

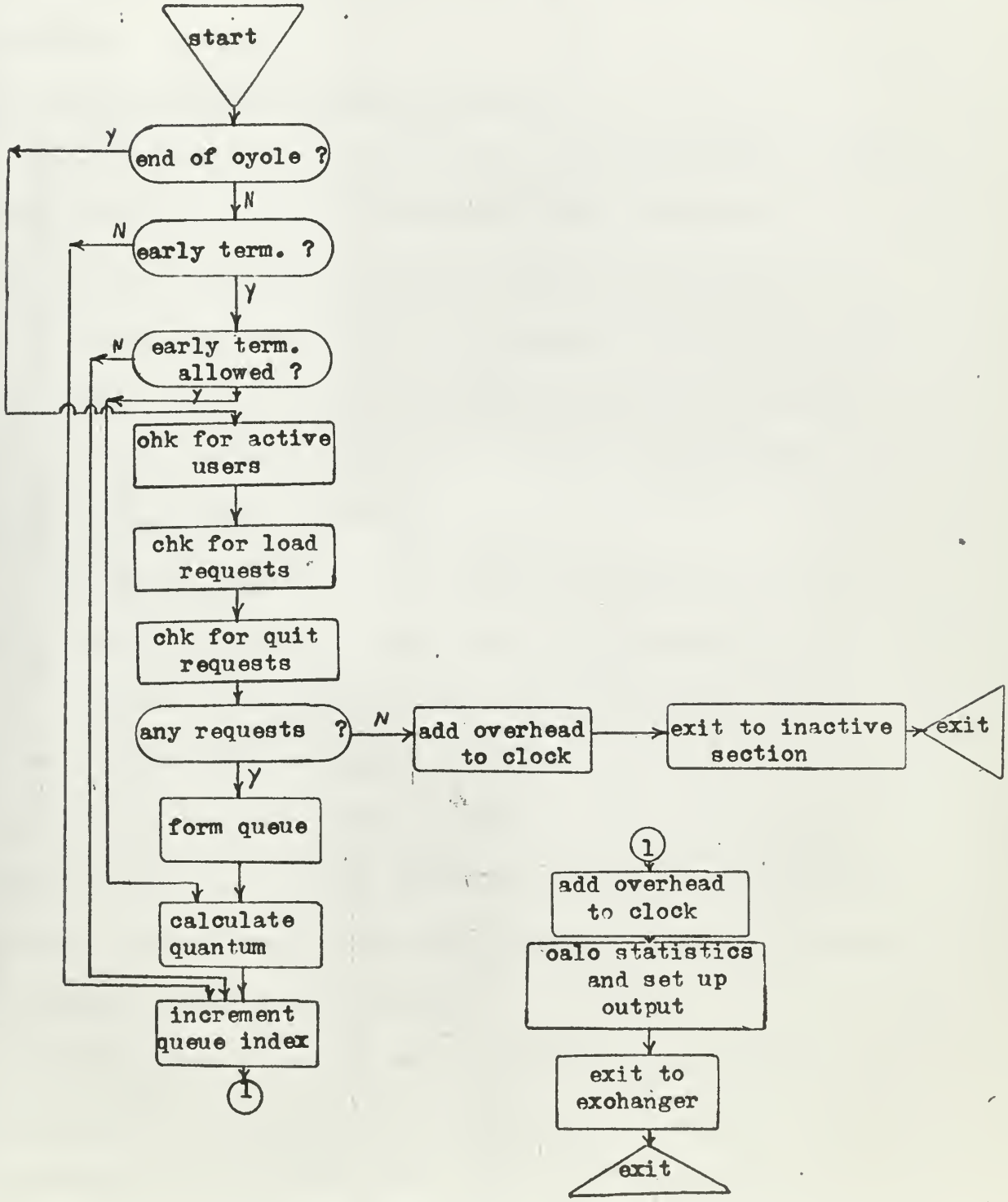
The variations to be investigated are:

(1) Not allowing an early termination of a program's allotted quantum to cause a recalculation and redistribution of the remaining response cycle time. Early termination means the termination of a job's active time by I/O operations or quitting before the normal quantum is complete.

(2) Permitting background user type jobs to be run when nothing else is active.

(3) Varying the maximum number of user stations allowed in order to determine the best level of operation.

FIGURE 5
Flow diagram of the scheduler section
of the simulator



(4) Varying the rate of arrival of job requests by varying the arrival time average in order to check on the overload conditions.

The variations will be brought about by both changes to the algorithm and changes to the input data.

3.4.2.3 Effects of Early Quantum Termination

In order to analyze this problem, the simulator was used to simulate a one hour normal run of the "Time-Sharing System" with its normal type of user activity. A one hour run each was made both with early terminations allowed to cause redistribution of remaining cycle time and without this capability. The input data was the same for each run and therefore each run was made within the exact same user request environment. An example of the job data is contained in Figure 6.

As the goal for Algorithm A is to give the best possible service to as many users as possible, the output data must be analyzed with this in mind. The run that accomplishes this goal best will determine the variation to be recommended for acceptance.

The first line of each section of Figure 7 is the output data from simulator run one where early termination has control over redistributing cycle time, while the second line of each section is from run two, where early termination does not have control.

An examination of the data contained in Figure 7 shows that run one honored more service requests, gave a larger average quantum and had a better computational efficiency than run two. The only advantage run two had over run one was the decrease in average scheduler overhead, which

FIGURE 6

GENERATED JOB DATA

JOB NUMBER	CLOCK TIME	STATION NUMBER	JOB TYPE	ARRIVAL TIME	LOAD TIME	ACTIVE TIME	I/O TIME	REPEATS	SIZE
1	.0	1	2	221.4191	2.0911	2.2392	1.9071	1.7934	7704
2	.0	2	3	422.6169	3.7595	3.1875	2.7417	3.9167	9967
3	.0	3	2	18.6029	2.2474	2.0644	2.4196	2.3021	7758
4	.0	4	4	51.7977	4.9365	3.3664	3.1046	3.7021	17001
5	.0	5	2	955.1382	2.3961	1.9972	1.9000	1.7958	7348
6	.0	6	4	224.1341	4.1193	5.0487	5.1255	3.3365	11518
7	.0	7	4	1182.1547	4.7424	4.9465	3.4368	4.1738	18690
8	.0	8	2	836.7463	2.1994	1.7704	2.5069	2.2042	7130
9	.0	9	3	112.8370	2.7981	2.7377	2.8158	2.1735	20053
10	.0	10	2	83.6587	1.5332	2.1563	1.9274	2.1941	7451
11	.0	11	1	27.7549	1.2320	1.0433	1.0808	1.1589	8272
12	.0	12	1	248.8739	1.1603	1.0866	.9582	1.1409	10774
13	.0	13	2	152.3819	1.8972	2.2984	1.5327	2.5686	7435
14	.0	14	4	5.3437	5.6259	3.2426	4.0263	4.4772	16774
15	.0	15	2	23.3176	2.3073	1.3500	1.7274	2.2705	9127
16	.0	16	1	39.4587	1.0351	.8134	.7934	.8366	8846
17	.0	17	3	427.2511	3.1780	3.0549	2.8314	3.5589	13534
18	.0	18	1	363.3659	.9843	.8398	1.0587	1.2013	10044
19	.0	19	3	209.7810	3.7059	3.3809	3.2497	2.3855	19469
20	.0	20	1	406.3655	1.0256	.8909	1.0085	.8926	7020
21	.0	21	2	191.4411	2.0834	2.1645	1.6389	1.8921	6822
22	.0	22	3	376.2925	2.8715	3.2557	3.5546	2.8507	18555
23	.0	23	4	38.3107	5.2660	3.5702	4.9525	3.7264	16307
24	.0	24	3	341.1849	2.9715	2.7980	3.0624	2.2100	14405
25	.0	25	2	192.6655	2.3799	1.2420	1.9678	1.8800	8559
26	.0	26	1	380.0252	1.1052	1.1800	.9964	.9065	8411
27	.0	27	4	306.0578	3.6862	4.2778	3.4938	4.9407	16592
28	.0	28	2	662.5954	2.3073	2.1391	2.0155	1.8192	8208
29	.0	29	4	202.2087	4.2518	3.8808	4.4473	2.6864	12179
30	.0	30	3	176.2448	3.8712	3.9903	2.7899	3.2730	20920

FIGURE 6 (CONTINUED)

31	.0	31	4	380.2991	4.0609	3.9923	3.2679	4.0264	16354
32	.0	32	1	354.8675	.8935	.8176	1.0613	1.3395	7033
33	.0	33	2	274.7550	2.1669	2.1419	1.3586	1.7944	8374
34	.0	34	3	391.4266	1.6410	3.7901	2.6151	2.9364	17279
35	.0	35	3	312.2573	3.1491	2.4795	2.5244	3.4310	18232
36	.0	36	2	421.6561	2.0825	1.8341	2.0174	2.3977	7584
37	.0	37	2	164.2482	2.1250	2.6811	2.1615	2.1623	9588
38	.0	38	4	416.8270	5.0705	4.0394	3.8063	3.8776	12675
39	.0	39	2	574.5804	1.7244	1.6553	1.9147	2.2912	7357
40	.0	40	2	464.6501	2.1209	2.1369	2.5862	1.1828	7446
41	.0	41	4	201.7328	2.5205	3.7198	4.4962	4.1008	12868
42	.0	42	1	319.0909	1.0068	.9705	.8517	.7201	7512
43	.0	43	4	815.5426	4.6437	5.1390	4.7397	3.5551	14300
44	.0	44	1	774.8624	1.0223	1.0402	.9268	1.1202	6630
45	.0	45	3	55.2453	3.6683	2.0454	3.5764	2.3529	15275
46	.0	46	2	387.5146	1.3806	1.8269	2.0054	2.3203	7791
47	.0	47	2	833.8090	2.0172	2.3712	2.0728	1.7160	7108
48	.0	48	3	21.6515	3.3418	2.7025	3.0900	2.8962	18647
49	.0	49	4	177.9286	3.8585	4.6074	5.2568	3.1892	13967
50	.0	50	2	38.4039	2.4257	2.0206	1.8718	1.7497	6603
51	109.5	16	1	142.3799	.8013	.7436	.8445	1.4489	6086
52	129.9	11	4	154.7163	4.5528	5.3597	3.2169	5.0286	16428
53	240.1	15	4	276.5654	4.1295	4.1511	4.0459	4.9232	13758
54	273.2	16	2	483.3657	1.7777	1.4752	1.4637	2.2587	8973
55	395.6	3	1	720.8634	1.2078	.7066	.9100	1.0108	7116
56	474.7	12	1	515.0353	1.4398	1.0079	1.3016	.9294	9451
57	554.9	18	1	1393.6008	1.3203	1.0662	1.1235	.9508	9518
58	592.1	10	2	1020.8646	2.1486	2.4023	1.3359	2.0294	8283
59	594.9	20	2	767.8813	2.2397	1.8648	2.3024	2.3709	8727
60	667.4	12	3	1467.6524	2.5943	4.0880	3.1847	3.5353	18035
61	684.4	13	4	1424.0239	4.4088	3.5847	3.6704	4.7690	14817
62	768.6	1	3	795.0603	3.1357	3.5561	2.5438	2.9961	14146
63	793.9	14	4	1195.2810	5.5359	5.2143	5.0344	4.8511	18132
64	826.9	9	3	869.5595	2.8087	3.2367	3.2336	3.2884	19142
65	837.2	16	3	973.0269	2.0252	3.5315	2.8885	3.1141	18635
66	893.4	3	2	951.7265	2.3172	2.0466	1.5161	1.7023	7145
67	1082.7	19	3	1469.3821	3.4191	2.9957	3.8120	3.3772	15505
68	1102.8	4	2	1948.2625	2.0067	1.8345	1.6482	2.2810	6713
69	1320.0	8	2	1770.3654	2.2592	2.8775	1.5411	2.4786	9407
70	1344.3	17	2	1418.8863	1.7948	1.8066	1.8362	1.9407	8618

FIGURE 6 (CONTINUED)

71	1433.5	20	2	1830.0501	2.0410	2.1324	1.3319	1.7532	7966
72	1443.7	5	2	2044.0764	2.2698	1.7091	1.8718	2.1329	8408
73	1482.3	3	2	2010.3982	2.5686	2.3252	1.3559	2.0923	7871
74	1569.9	18	1	1684.9348	.8897	.5650	.9018	.8077	9252
75	1728.3	10	3	1816.4442	3.1112	3.6246	2.9923	3.1798	15947
76	1758.4	18	1	2007.3856	1.2438	1.0816	.9612	1.0989	6872
77	1762.5	2	3	2018.9561	2.4027	2.8321	3.2339	2.9617	14578
78	1809.3	15	4	1816.4295	4.5706	4.0135	3.9089	2.5230	16957
79	1817.3	17	4	2145.2779	4.1831	3.7771	3.6139	3.5784	13401
80	2108.6	9	4	2239.0568	2.5982	3.2554	3.3629	4.2593	18989
81	2131.8	1	1	2342.3738	.6298	.8137	1.2273	.9177	9752
82	2191.2	16	3	2267.7441	2.4547	3.7969	3.3267	2.9322	16125
83	2193.7	6	4	2325.7695	3.4519	4.6203	4.7190	3.9960	13823
84	2286.6	18	3	2402.1645	2.6568	2.7332	4.5108	2.4300	19075
85	2569.9	1	2	2716.8538	2.1192	1.8831	2.2533	1.7075	7033
86	2634.5	20	4	2845.7480	4.6392	4.0904	4.6612	2.6685	16047
87	2783.7	4	1	2966.0950	1.2027	.8149	.9952	.8334	8695
88	2801.9	5	4	3051.3215	5.7952	2.1376	3.2282	2.4819	13501
89	2863.6	19	4	3485.5317	2.6303	4.6973	4.1103	3.6266	15901
90	2899.3	11	1	3536.4650	1.1444	1.3459	.9510	1.0127	8002
91	2940.5	8	2	3107.5460	2.0919	1.9058	2.0652	1.7831	6651
92	3073.2	3	3	3310.2998	3.9072	3.2406	2.1693	3.3698	15033
93	3143.1	15	4	3157.3545	4.7646	3.2084	3.2391	4.5540	13761
94	3151.5	4	4	3499.8593	4.7897	4.1860	4.4912	3.3031	16180
95	3241.0	13	1	3326.1392	1.2168	1.0799	1.2717	1.1807	7828
96	3245.7	12	1	3820.3796	1.2658	.9863	1.4298	.8448	9583
97	3290.9	18	3	3434.6265	2.9782	2.9367	3.8876	3.2036	17658
98	3302.1	2	4	3460.4591	3.0808	3.2467	4.2285	3.6832	20972
99	3342.4	1	2	3570.7548	1.7845	1.8035	1.8501	1.7013	7477
100	3379.8	16	4	3481.5638	4.6352	3.3311	5.0490	5.5792	15919
101	3430.6	10	3	4149.6411	2.9069	2.3853	3.2587	2.7184	18222
102	3538.5	17	4	4750.0147	5.1100	4.8400	4.5267	3.0083	11794

FIGURE 7

SIMULATOR OUTPUT DATA
WITH EARLY TERMINATIONS ALLOWED,
NOT ALLOWED AND BACKGROUND JOBS PERMITTED

VARIABLE PARAMETER	CYCLE COUNT	AVERAGE CYCLE TIME	AVERAGE NUMBER IN QUEUE	AVERAGE QUANTUM	AVERAGE OVERHEAD	COMPUTATIONAL EFFICIENCY.	AVERAGE EXCHANGE OVERHEAD	AVERAGE EXCHANGE TIME
1.0000	1765.0000	2.3142	13.2312	.0460	.0566	26.3198	.0035	.9634
2.0000	1534.0000	2.3403	14.0202	.0289	.0120	17.3100	.0042	1.1615
3.0000	1750.0000	2.3637	13.5640	.0464	.0576	26.6178	.0036	.9653

AVERAGE OVERLOAD	MAXIMUM QUANTUM	MAXIMUM CYCLE TIME	MAXIMUM NUMBER IN QUEUE	MAXIMUM OVERHEAD	MAXIMUM OVERLOAD	MAXIMUM NUMBER STATIONS	STATIONS SERVICED
.0006	.2000	8.3686	20.0000	.9720	1.0000	20.0000	69.0000
.0007	.2000	8.4429	20.0000	.0160	1.0000	20.0000	59.0000
.0000	.2000	8.4705	20.0000	.9320	.0000	20.0000	66.0000

was offset by the fact that the scheduler overhead represents but a small part of the total overhead (scheduler overhead, exchanger overhead and exchange time) and therefore the total decrease was negligible.

The outcome of the simulator analysis clearly shows that the variation of the algorithm that allows early termination to cause a redistribution of remaining response cycle time appears to give better service to the users and therefore should be implemented within Algorithm A.

3.4.2.4 Effects of Allowing Background Users

A simulated one hour run (run one) was made by the simulator without background users and another (run three) was made with background users in order to determine if background users should be allowed or not. Lines one and three of each section of Figure 7 represent the simulator output for runs one and three respectively.

The data shows that, while the number of service requests honored while allowing background users was not as high as while not allowing them, the reduction was small and the computational efficiency went up. That the computational efficiency went up only slightly, is in part due to the system already being used very efficiently (for a multiprogramming system) and therefore not leaving much inactive time for the background user to take advantage of.

It is recommended that background users be allowed, even though the efficiency gain is slight, in order that large, long computing jobs (such as compilations) may be run as background and not degredate the other users' service as much as they would if they were run as regular time-sharing jobs.

3.4.2.5 Effects of Varying Maximum Number of Users

To study the effect of variations in the number of users allowed, ten one hour runs were simulated. The number of users for each run was incremented by five at the completion of each run. The range of users allowed was from 5 to 50.

As the variable parameter was the number of users, the number for each run may be found in the variable parameter column of the first section of Figure 8 as well as in the maximum number of stations column of the second section of Figure 8.

As was expected, the results of the simulation study showed that, as the number of user stations allowed went up, the total overhead went up, the computational efficiency went down and the response time (average cycle time) increased.

Due to the advisability of maintaining a response time close to the human reaction time (2 seconds), the simulator results show that a limit of 20 should be set on the number of user stations allowed.

3.4.2.6 Effects of Varying the Rate of Arrival of Service Requests

The varying of the rate of arrival of service requests was accomplished by varying the average or mean used to generate the times of arrival of requests, due to the fact that if the mean is lowered, while the number of jobs generated remain the same, the rate of arrival of jobs will increase. The mean was varied from 300 to 50 seconds. An hour was simulated for each new mean. The first run was made with a mean arrival time of 300 seconds and each succeeding run with a mean of 50 seconds less than the preceding one.

FIGURE 8

SIMULATOR OUTPUT DATA
WITH NUMBER OF USERS VARIABLE

VARIABLE PARAMETER	CYCLE COUNT	AVERAGE CYCLE TIME	AVERAGE NUMBER IN QUEUE	AVERAGE QUANTUM	AVERAGE OVERHEAD	COMPUTATIONAL EFFICIENCY	AVERAGE EXCHANGE OVERHEAD	AVERAGE EXCHANGE TIME
5.0000	3388.0000	.4048	1.5611	.1587	.0130	61.1879	.0005	.0886
10.0000	4107.0000	.8030	4.3630	.0756	.0168	41.0901	.0013	.3401
15.0000	2443.0000	1.6072	9.1793	.0415	.0334	23.6880	.0025	.7035
20.0000	1753.0000	2.3532	13.3896	.0462	.0599	26.2674	.0035	.9747
25.0000	1438.0000	2.9536	16.8978	.0486	.0889	27.7996	.0043	1.1756
30.0000	1367.0000	3.1383	17.9920	.0481	.0959	27.5534	.0045	1.2294
35.0000	1355.0000	3.1650	18.0421	.0480	.1000	27.3878	.0045	1.2436
40.0000	1353.0000	3.2023	18.1057	.0475	.0997	26.8594	.0046	1.2501
45.0000	1332.0000	3.2449	18.3461	.0483	.1036	27.3039	.0046	1.2580
50.0000	1318.0000	3.3086	18.7853	.0474	.1124	26.8919	.0047	1.2672

FIGURE 8 (CONTINUED)

AVERAGE OVERLOAD	MAXIMUM QUANTUM	MAXIMUM CYCLE TIME	MAXIMUM NUMBER IN QUEUE	MAXIMUM OVERHEAD	MAXIMUM OVERLOAD	MAXIMUM NUMBER STATIONS	STATIONS SERVICED
.0000	.2000	5.8574	4.0000	.1720	.0000	5.0000	52.0000
.0000	.2000	6.8921	9.0000	.3720	.0000	10.0000	61.0000
.0004	.2000	7.6677	15.0000	.6720	1.0000	15.0000	61.0000
.0017	.2000	8.4923	20.0000	.9220	1.0000	20.0000	67.0000
.7752	.2000	9.1665	20.0000	.9820	5.0000	25.0000	74.0000
4.0037	.2000	9.4893	20.0000	1.0720	9.0000	30.0000	75.0000
7.8789	.2000	9.2676	20.0000	1.0220	14.0000	35.0000	76.0000
12.1783	.2000	9.1303	20.0000	1.0720	19.0000	40.0000	74.0000
16.0225	.2000	9.2020	20.0000	1.0220	24.0000	45.0000	79.0000
20.7555	.2000	9.0836	20.0000	1.1220	29.0000	50.0000	77.0000

The output of the simulator analysis, as shown in Figure 9, indicates that the overload increases as the rate of arrival increases with a corresponding decrease in computational efficiency.

It is very apparent that the rate of arrival of service requests greatly influences the loading of the system. Fortunately, except for an initial surge when the system begins operation for the day, a multiprogramming system has a fairly low request arrival rate due to the slowness of the human thought and action process.

3.4.3 General Summary of Simulation Analysis

The example of the simulator analysis of Algorithm A illustrated the flexibility and effectiveness of simulation analysis and outlined the method of using the simulator.

The simulator may be used to analyze and compare several different scheduling algorithms in exactly the same manner it was used to analyze and compare several variations of Algorithm A.

FIGURE 9

SIMULATOR OUTPUT DATA
WITH ARRIVAL TIME AVERAGE VARIABLE

VARIABLE PARAMETER	CYCLE COUNT	AVERAGE CYCLE TIME	AVERAGE NUMBER IN QUEUE	AVERAGE QUANTUM	AVERAGE OVERHEAD	COMPUTATIONAL EFFICIENCY	AVERAGE EXCHANGE OVERHEAD	AVERAGE EXCHANGE TIME
4.0000	1366.0000	3.1508	17.9949	.0486	.1049	27.7299	.0045	1.2383
4.0000	1240.0000	3.5287	19.7774	.0463	.1058	25.9672	.0049	1.3781
4.0000	1236.0000	3.4971	19.7581	.0464	.1057	26.2264	.0050	1.3978
4.0000	1241.0000	3.5022	19.7744	.0473	.1080	26.7185	.0049	1.3782
4.0000	1237.0000	3.5171	19.7704	.0466	.1068	26.1977	.0049	1.3878
4.0000	1253.0000	3.4691	19.7414	.0472	.1084	26.8863	.0049	1.3649

FIGURE 9 (CONTINUED)

AVERAGE OVERLOAD	MAXIMUM QUANTUM	MAXIMUM CYCLE TIME	MAXIMUM NUMBER IN QUEUE	MAXIMUM OVERHEAD	MAXIMUM OVERLOAD	MAXIMUM NUMBER STATIONS	STATIONS SERVICED
3.9758	.2000	9.7255	20.0000	5.3560	9.0000	30.0000	75.0000
6.4746	.2000	9.0557	20.0000	1.0720	10.0000	30.0000	75.0000
7.3093	.2000	9.2733	20.0000	1.0220	10.0000	30.0000	71.0000
7.7766	.2000	9.2392	20.0000	1.0720	10.0000	30.0000	73.0000
8.2290	.2000	9.7907	20.0000	1.0720	10.0000	30.0000	74.0000
9.0136	.2000	9.1523	20.0000	.9720	13.0000	30.0000	73.0000

4.0 Conclusions and Recommendations

An investigation of several different methods of analyzing the operation of a multiprogramming scheduling algorithm was conducted in section 3.1. The advantages and disadvantages of each method were determined and a composite investigation procedure was proposed that would fully utilize the various favorable characteristics of each method and negate the unfavorable ones. This proposed investigation procedure, as outlined in section 3.2, was demonstrated by the example of the investigation of Algorithm A in sections 3.3 and 3.4.

As shown by the example, the use of the composite investigation procedure will bring to light many important facts concerning the operation of a scheduling algorithm and allow changes to be made to an algorithm or a new one adopted without disrupting the system operation. The use of the investigation procedure insures that the scheduler investigated will bring about a close realization of the system's goals.

The technique of simulation was shown to be a very powerful and versatile tool for use in the analysis of an algorithm's operation. The simulator used for the investigation of the operation of Algorithm A demonstrated these qualities by its detailed portrayal of a wide variety of system parameters of interest during the simulated operation of the system. One additional feature of the simulator, that was noticed during the course of the investigation, is the detailed insight obtained of not only the operation of the scheduler, but also the complete multiprogramming system. This insight was gained by the process of

formulating and coding the simulator. This experience and knowledge of the system would be invaluable to the system's programmer and a simulation study of a system will pay off in this respect alone.

Some of the major problems involved in combining the on-line control type user with the normal users of a multiprogramming system were outlined during the heuristic analysis of the algorithm proposed by the author (Algorithm C) in section 3.3.3. The foremost obstacle being the inability of the control type user to allow his service requirements to be varied, in order to be compatible with the rest of the users. It must be accepted that all other users bow to the requirements of the control user i.e., the control user's requirements in effect determine the scheduler design. With this in mind, Algorithm C and the modifications to it, as proposed in section 3.3.3.4, offer reasonable solutions to the problem of integrating the control type user into the multiprogramming system.

The state of the art in the multiprogramming area may be briefly described as just coming out of the hardware limited phase. Heretofore it has been extremely difficult to obtain the necessary capabilities in a computer system to allow effective multiprogramming. As a result, many features that should have been of a hardware nature have, in some cases, been implemented by programming (with a resulting increase in overhead) or have been omitted. A few of such features are:

(1) Priority assignment of interrupts and users.

(2) Quantum determination and interruption.

(3) Memory protection.

With the new large computers being designed for multiprogramming applications, this state of affairs will cease to exist and the simple scheduling algorithms of today will have to be replaced by more complex and versatile ones. The algorithms in present use are entirely satisfactory for the present requirements of 20 to 50 user stations utilizing Teletypes and/or displays as I/O devices and operating in such a manner as to require only a small portion of their time in the system to be compute time.

The present scheduling philosophy is capable of providing algorithms for use within systems, that concern themselves primarily with satisfying a fairly large number of users, whose needs are short bursts of compute time, followed by longer intervals of idle or I/O time. A change in philosophy, however, appears to be necessary in order to provide adequate scheduling algorithms for use within the military command and control type of multiprogramming system, where the numerous tasks and service requirements are both extremely complex and exacting. This change also appears necessary when considering the type of system that contains multiple facilities (such as multiple processing units, etc.) and requires the scheduler to schedule the use of each and every one of the multiple units. These problems are too complex for solution by present scheduler philosophy and represent areas of multiprogramming that are fertile fields for future research.

Further research into the use of simulation techniques for analysis of scheduling algorithms is needed as the capabilities of simulation are

enormous, if utilized properly and integrated with other techniques in the correct manner.

A final recommendation is that the applicability, of dynamic programming to the scheduler problem, be investigated thoroughly, as the analysis of a scheduler is a form of cost-effectiveness study and dynamic programming is very effective in this area.

BIBLIOGRAPHY

1. Bright, H. S., and B. F. Cheydeur; "On The Reduction of Turn-around Time"; Proceedings of The Eastern Joint Computer Conference, 1962.

2. Codd, E. F.; "Multiprogram Scheduling"; Communications of The Association for Computing Machinery, June 1960 and July 1960.

3. Codd, E.F.; "Multiprogramming"; Advances in Computers, Volume 3, 1962.

4. Corbato, F. J., M Merwin-Daggett, and R. C. Daley; "An Experimental Time-Sharing System"; Proceedings of The Spring Joing Computer Conference, 1962.

5. Corbato, F. J. and others; "The Compatible Time-Sharing System. A Programmer's Guide"; Cambridge, Massachusetts, M.I.T. Press, 1963.

6. Critchlow, A. J.' "Generalized Multiprocessing and Multi-programming Systems"; Proceedings of The Fall Joint Computer Conference, 1963.

7. Fredkin, E.; "The Time-Sharing of Computers"; Computers and Automation, Volume 12, November 1963.

8. Goode, H. H., and R. E. Machol; "System Engineering"; McGraw-Hill, 1957.

9. Howarth, D. J.' "Experience With The Atlas Scheduling System"; Proceedings of The Spring Joint Computer Conference, 1963.

10. Kilburn, T., R. B. Payne, and D. J. Howarth; "The Atlas Supervisor"; Proceedings of The Eastern Joint Computer Conference, 1961.

11. Licklider, J. C. R., and W. E. Clark; "On-Line Man-Computer Communication"; Proceedings of The Spring Joint Computer Conference, 1962.

12. Morse, P. M.; "Queues, Inventories and Maintenance"; John Wiley and Sons, 1958.

13. McKenney, J. L.' "Simultaneous Multiprogramming of Electronic Computers"; University of California, Los Angeles, February 1961.

APPENDIX I

SIM-A MULTIPROGRAMMING SYSTEM SIMULATOR

Simulation has become a valuable analytic method that can be applied in diverse situations. The type of simulation of interest is the Monte Carlo Method which is defined in the McGraw Hill Encyclopedia of Science and Technology, as "A technique for estimating the solution, x, of a numerical mathematical problem by means of an artificial sampling experiment . . ." The method aptly fits the multiprogramming system problem and can produce worthwhile results. The required probability distributions associated with users can be determined by general data gathering and observation. The use of various algorithms in the Executive routine and several hardware configurations in a simulated system subjected to a typical loading will produce the data to obtain a measure of effectiveness for the various hardware and software configurations. The time and expense required to actually evaluate each of the combinations in an operating system are prohibitive and the use of simulation techniques provides the only realistic approach.

Program SIM was developed as a general multiprogramming system simulator with the emphasis on the time-sharing type of environment. Due to the specific nature of the authors' theses, primary attention was given to the Scheduler and Exchanger. The normal performance of other specific areas of the Executive routine is assumed and these portions treated in a block method. A prime example of this is the Dispatcher. While it is a critical area of a multiprogramming system no specific

characteristics were assumed. When a user has an Input or Output the program assumes a waiting status until, due to the incrementing of the simulator clock, the action is deemed complete. The availability of the required I/O equipment at all times is assumed. A time-sharing system is characterized by frequent man-machine communication and buffered I/O is usually impossible due to the step by step nature of the system. However, simultaneous I/O by all users, at least to their reactive typewriters, must be permitted. The gross Dispatcher treatment provides all this and only avoids the complications of particular operations. If this area is studied in the future the Dispatcher portion could easily be made more detailed and added to the system simulator.

The job load on the simulated system is created by a job generation subroutine (SET). Each job is characterized by six variables, which define any job entering the system. Arrival time, the first parameter, is assumed to be exponentially distributed on the basis of queuing theory concepts and actual observation at System Development Corporation. A variable parameter is the mean arrival time expressed in seconds. The value of arrival time was determined by taking the natural logarithm of a uniformly distributed random number. The second parameter is Load time and represents the time required to transfer the binary program from its permanent storage to the temporary storage having access to the central memory. The next three parameters, Active time, I/O time and Repeats define the actual program operating characteristics. A program once loaded into the system is assumed to have an active period followed by

an I/O period, during which no service is required from the central processor. This cycle is repeated until the job is completed or there are no further repeats required. Due to the nature of SIM, differences in I/O such as tape transfers, searches or outputs to reactive devices are not recognized and the I/O operations are grouped together. The sixth parameter, Size, completes the job description. Program size is limited to a maximum length of one hundred cells less than the full core available for operating programs. The last five parameters are determined using a Gaussian Random number generator and the mean values received as input. A uniform random number generater is used to generate any of ten possible job types. The probability of each type job is also received as an input. As soon as the job is completed, that is the number of repeats remaining is less than zero, a new job is generated for that station. An example of the input to SIM is shown at the end of the program contained in Appendix II.

It is possible to obtain a wide variety of output parameters from the simulator as it has access to all of the internal system parameters concerning the operation of the system in question. These parameters may be gathered on a minute, average, total or maximum basis and thereby present a picture of the system's operation in almost any degree of detail desired.

For the purpose of comparison, the output parameters of one run, using a certain hardware and software configuration, may be saved and then presented with the results of another run, with changes to the

hardware and/or software configuration. The output of the simulator may be in a tabular format or modified to a graphical type format, which ever is deemed best for comparison purposes.

The program operation is cyclic in nature. First, the initial jobs are generated and the program constants read in and/or initialized for the run. The main body of the simulator is then entered and the actual run commenced. The clock is checked against arrival times of all allowed users (maximum 50) and an equality or greater than condition sets the action entries of the status table (STAT(X,Y)). The Scheduler then determines which requests for service shall be honored and the order in which they shall be honored, i.e., queue formation. The Scheduler also determines, from the number requesting service, the amount of time each user is allowed per cycle. The cycle begins with the formation of the queue and ends with termination of the last user's quantum. The Scheduler then passes control to the Exchanger. For further specific information concerning the operation of the Scheduler section of the simulator see Lt. W. G. Wilder's thesis.

The Exchanger determines the action required by the next user in the queue and then LOADS, QUITS or TRANSFERS the users program. The actual transfer algorithm is variable and the methods used are discussed in detail in Lt. R. R. Hatch's thesis. Regardless of the exact method, the required transfers are determined and the effective transfer times (TELOAD and TEDUMP) and exchange overhead calculated and added to the clock. In the LOAD and QUIT operations the size of

the external store, such as a drum, is considered and no load or storage full conditions are possible.

At the completion of a cycle all users in an I/O status are handled by decrementing the remaining time by the elapsed cycle time. Users completing I/O are checked for repeats. If repeats are necessary the program is reset to the active mode and if no repeats are required the program is terminated by a QUIT command in the next cycle.

To avoid long idle periods a scheme is used to advance the clock to the next active clock time if there are no users desiring service. The smallest I/O time remaining (SMALLA) and the nearest arrival time (SMALLB) are determined and the smaller of these two added to the clock and a new cycle commenced.

A maximum clock parameter read in terminates the run and the capability for recycling is provided. All new parameters may be read in or the original parameters may be modified for successive runs.

A flow diagram of SIM is contained in Figure I-1 and a copy of the actual program is contained in Appendix II.

Due to the fact that the thesis topics of Lt. W. G. Wilder and Lt. R. R. Hatch are both in the multiprogramming area and a generalized multiprogramming simulator could be used in each case, this simulator represents the joint efforts of both Lt. Wilder and Lt. Hatch.

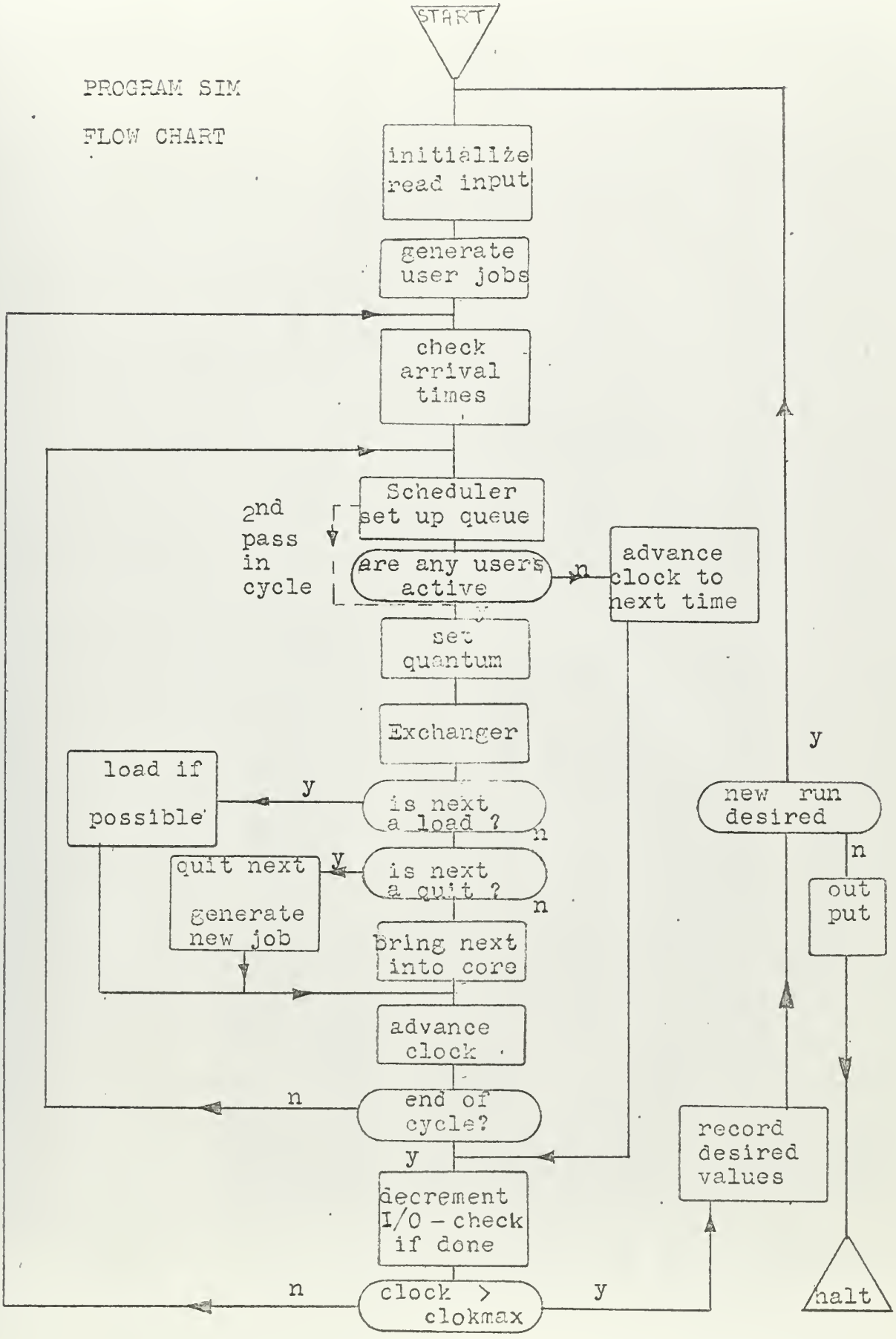

START
PROGRAM SIM
FLOW CHART
initialize read input
generate user jobs
check arrival times
Scheduler set up queue
2nd pass in cycle
are any users active
n
advance clock to next time
set quantum
Exchanger
load if possible
y
is next a load ?
n
quit next generate new job
y
is next a quit ?
n
bring next into core
advance clock
n
end of cycle?
y
decrement I/O - check if done
n
clock > clokmax
y
halt
y
new run desired
n
out put
record desired values

FIGURE I-1

APPENDIX II

PROGRAM SIM

```
      DIMENSION SAVE(50,5)
      DIMENSION MCORE(50,4) , MDRUM(50,4) , MDISC(50,4)
      DIMENSION IQUIT(50),IQUE(50),GENR(50,10),STAT(50,15),
     1AVERAGE(10,8),OUTABL(15),INPUT(40),DIST(10),DATA(10),TOTOUT(30,50)
      COMMON GENR,AVERAGE,DIST,JN,JOBTYPE,CLOCK,DST,ICUTCON
      COMMON MAXDRM,MAXDSC,MAXCOR,IR
      COMMON GAUSRN, UNIRN
    1 FORMAT(7F10.3)
    6 FORMAT (7F10.3)
    7 FORMAT(6I2)
    9 FORMAT(7F10.3)
   50  FORMAT(4I6)
   51  FORMAT(1H1)
   52  FORMAT(////)
   54  FORMAT(30X,32HWITH EARLY TERMINATIONS ALLOWED,)
   55  FORMAT(25X,41HNOT ALLOWED AND BACKGROUND JOBS PERMITTED//)
   60  FORMAT(/////////)
   80  FORMAT(//2X,8HVARIABLE,5X,15HCYCLE    AVERAGE,3X,7HAVERAGE,3X,
     17HAVERAGE,3X,21HAVERAGE COMPUTATIONAL,8H AVERAGE,2X,7HAVERAGE)
   81  FORMAT(10H PARAMETER,5X,14HCOUNT     CYCLE,4X,6HNUMBER,4X,7HQUANTUM
     1,2X,19HOVERHEAD EFFICIENCY,3X,8HEXCHANGE,2X,8HEXCHANGE)
   82  FORMAT(24X,4HTIME,4X,8HIN QUEUE,34X,8HOVERHEAD,3X,4HTIME//)
   83  FORMAT(//2X,7HAVERAGE,4X,7HMAXIMUM,3X,7HMAXIMUM,2X,7HMAXIMUM,3X,
     17HMAXIMUM,3X,7HMAXIMUM,3X,7HMAXIMUM,3X,8HSTATIONS)
   84  FORMAT(2X,8HOVERLOAD,3X,7HQUANTUM,3X,5HCYCLE,4X,6HNUMBER,4X,
     18HOVERHEAD,2X,8HOVERLOAD,2X,6HNUMBER,4X,8HSERVICED)
   85  FORMAT(23X,4HTIME,5X,8HIN QUEUE,22X,8HSTATIONS//)
   86  FORMAT(//2X,3HJOB,4X,20HCLOCK   STATION  JOB,4X,7HARRIVAL,3X,
     14HLOAD,6X,6HACTIVE,9X,3HI/O,5X,7HREPEATS,5X,4HSIZE)
   87  FORMAT(6HNUMBER,3X,4HTIME,4X,12HNUMBER  TYPE,4X,4HTIME,11X,4HTIME
     16X,4HTIME,10X,4HTIME//)
   88  FORMAT(//40X,8HFIGURE 7///)
   89  FORMAT(35X,21HSIMULATOR OUTPUT DATA)
   90  FORMAT(2F15.3,F10.6,3F15.1)
   91  FORMAT(F15.1,2F6.1,F12.6,2F9.1)
   92  FORMAT(10F9.6)
```

```
   93  FORMAT(//44X,8HFIGURE 6//)
   94  FORMAT(39X,18HGENERATED JOB DATA////)
  515  FORMAT(9F10.4//)
  516  FORMAT(8F10.4//)


       IR=1
       ID=0
       IE=0
       NREADF=0
       PRINT 51
       PRINT 60
       PRINT 93
       PRINT 94
       PRINT 86
       PRINT 87
       READ 1,((AVERAGE(J,I),I=1,7 ),J=1,10)

C      INITIALIZE

    10 ENDCYCL=1.0
       JN = 0
       JP=0
       JD=0
       JC=0
       UNIRN = 0.374821
       INP=0
       IF(MEHLT)15,12,15
    12 CONTINUE
       GO TO 14
    16 CONTINUE
       ISTOPF = 1
       GO TO 570
    15 CONTINUE
       PLENGTH = AVERAGE(I,6)
       AVERAGE(I,6)=PLENGTH + 1000.0
       AVERAGE(1,7) = 1.0
       IF(AVERAGE(1,6) - 64000.0) 14,14,16
    14 CONTINUE
       IF(NREADF)45,45,46
    45 CONTINUE
       READ 50, MAXDRM,MAXDSC, MAXCOR ,MEHLT
       READ 6,QA,QB,QC,QD,QE,QF,SOVRHD1
       READ 9,SOVRHD2,SOVRHD3,CLOKMAX,LMINQ
       READ 7,NCYCTM,ISTOPF,ISKEDTP,NUSERS,IVARY,IOUTCON
       NU=NUSERS
    46 CONTINUE
```

```
      JOBKIND=4
      DST=0.0
      CLOCK=0.0
      DO 3 J=1,JOBKIND
    3 DIST(J)=0.0
      DO 4 J=1,50
      DO 4 I=1,10
      GENR(J,I)=-1.0
    4 STAT(J,I)=-1.0

C     GENERATE FIRST 50 JOBS

      DO 5 J=1,50
    5 CALL SET(J)
      NDRUM=0
      SOVD = 0.0
      EXOVO = 0.000030
      EXOVF = 0.000100
      LDCHK=0
      NWCYCL=1
      DRUM=0.0
      CORET = 0.0
      WAITT = 0.0
      WAITC = 0.0
      CYCNT=0.0
      COREUT=0.0
      QTOT=0.0
      TCYTM=0.0
      TNQUE=0.0
      NOLOAD = 0
      TEFEXCH =0.0
      TEXCH = 0.0
      CSOVRHD=0.0
      SOVRHD=0.0
      SOVRHDM=0.0
      CYTIMMX=0.0
      MNQUE=0
      MNOVLD=0
      ITOVLD=0
      INU=1
      REQUEST=0.0

C     SCHEDULER SET-UP

      GO TO (20,21,22,23),IVARY
   20 VARPARA=MINQ
      GO TO 25
```

```
   21 VARPARA=ISKEDTP
      GO TO 25
   22 VARPARA=NCYCTM
      GO TO 25
   23 VARPARA=NUSERS
   25 CONTINUE
      FCYCTM=NCYCTM
      IQMAX=FCYCTM/EMINQ
      GO TO (30,31,32,33,34),ISKEDTP
C     REGULAR ALGORITHM
   30 INET=2
      GO TO 100
C     EARLY TERM. NOT ALLOWED
   31 INET=1
      GO TO 100
C     BACKGROUND USERS ALLOWED
   32 GENR(1,1)=CLOCK
      GENR(1,3)=CLOKMAX
      INU=2
      INET=2
      GO TO 100
C     ARRIVAL TIME AVERAGE VARIABLE
   33 DO 40 IB=1,10
      INET=2
      AVE=AVERAGE(IB,1)
      AVERAGE(IB,1)=AVE-50.0
      IF(AVERAGE(IB,1))41,41,40
   40 CONTINUE
      NREADF=1
      ID=ID+1
      IF(ID-10)42,42,41
   41 ISTOPF=1
      NREADF=0
      GO TO 570
   42 GO TO 100
C     NUMBER OF USERS VARIABLE
   34 NU=NU+5
```

```
      INET=2
      IF(IVARY-4)48,49,48
   49 VARPARA=NU
   48 CONTINUE
      NREADF=1
      IE=IE+1
      IF(IE-10)44,43,43
   43 NREADF=0
   44 GO TO 100

C     ARRIVAL DETERMINATION

  100 TIMERUN=0.0
      I=1
  101 IF(GENR(I,1)-CLOCK) 102,102,103
  102 STAT(I,1)=1.0
      STAT(I,2)=1.0
      STAT(I,9)=1.0
      GEN=GENR(I,1)
      GENR(I,1)=GEN+1000000.0
  103 IF(I-NU) 104,105,105
  104 I=I+1
      GO TO 101
  105 SCYCLE=CLOCK
      GO TO 1000

C     NO ACTIVE USERS

  400 CONTINUE
      ENDCYCL=1.0
      CYTIME=CLOCK-SCYCLE
      SMALLB = GENR(1,1) - CLOCK
      DO 401  I =2,NU
      IF(GENR(I,1) - CLOCK) 401,410,410
  410 CONTINUE
      IF(GENR(I,1) - CLOCK - SMALLB) 402,401,401
  402 SMALLB = GENR(I,1) - CLOCK
  401 CONTINUE
      SMALLA = STAT(1,5)
      IF(SMALLA) 403,404,404
  403 SMALLA = 1000000.0
  404 CONTINUE
      DO 407 I=2,NU
      IF(STAT(I,5)) 407,408,408
  408 CONTINUE
      IF(SMALLA - STAT(I,5)) 407,407,409
  409 SMALLA = STAT(I,5)
```

```
  407 CONTINUE
      IF(SMALLA-SMALLB)405,406,406
  405 CLOCK=CLOCK+SMALLA
      GO TO 360
  406 CLOCK=CLOCK+SMALLB
      GO TO 360

C     USER IS LOADING

  200 N=IQUE(NEXT)
      REQUEST=REQUEST+1.0
      LDCHK=0
      CLOCK=CLOCK+GENR(N,2)
      STAT(N,4)=GENR(N,3)
      GO TO 300

C     USER IS QUITTING

  210 I=IQUE(NEXT)
      STAT(I,1)=-1.0
      CALL SET(I)
      GO TO 300

C     USER IS ACTIVE

  220 N=IQUE(NEXT)
      IF(STAT(N,4)-Q)221,221,222
  222 ACTIME=STAT(N,4)-Q
      STAT(N,4)=ACTIME
      TIMERUN=Q
      CLOCK=CLOCK+Q
      GO TO 300

C     EARLY TERMINATION

  221 TIMERUN=STAT(N,4)
      TERM=1.0
      STAT(N,9)=-1.0
      STAT(N,5)=GENR(N,4)
      CLOCK=CLOCK+TIMERUN
      GO TO 300

C     END OF CYCLE CHECK AND HOUSE-KEEPING

  300 IF(ENDCYCL)1000,1000,350
  350 CYTIME=CLOCK-SCYCLE
  360 CONTINUE
```

```
      ACYTIME=CLOCK-SCYCLE
      DO 351 I=1,NU
      IF(STAT(I,5))351,351,352
  352 IF(STAT(I,10))358,358,357
  358 STAT(I,10)=1.0
      GO TO 351
  357 TIME=STAT(I,5)-ACYTIME
      IF(TIME)353,353,354
  354 STAT(I,5)=TIME
      GO TO 351
  353 CONTINUE
      STAT(I,5)=-1.0
      STAT(I,10)=-1.0
      IF(GENR(I,5))356,356,355
  355 STAT(I,4)=GENR(I,3)
      STAT(I,9)=1.0
      REPT=GENR(I,5)-1.0
      GENR(I,5)=REPT
      GO TO 351
  356 CONTINUE
      STAT(I,9)=1.0
      STAT(I,6) = 1.0
  351 CONTINUE
      IF(CLOCK-CLOKMAX) 100,100,500

C BASIC SCHEDULER

 1000 IF(ENDCYCL)1060,1060,1050
 1060 IF(TERM)1020,1020,1430
 1430 GO TO (1020,1040),INET

C     NEW CYCLE

 1050 IT = 1
      ENDCYCL=0.0
      IQ = 1
      NEXT=0
      CLOCK=CLOCK+SOVRHD3
      SOVRHD=SOVRHD+SOVRHD3

C     QUEUE FORMATION FIRST PHASE

      DO 1120 IS=INU,NU
      IF(STAT(IS,9))1120,1120,1070
 1070 IF(STAT(IS,2))1010,1010,1080
 1080 IF(LDCHK)1090,1090,1120
 1090 ILOD =IS
```

```
      LDCHK=1
      GO TO 1120
 1010 IF(STAT(IS,6))1100,1100,1110
 1110 IQUIT(IT)=IS
      IT=IT+1
      GO TO 1120
 1100 IQUE(IQ)=IS
      IQ=IQ+1
      IF(IQ-IQMAX)1120,1120,1440
 1120 CONTINUE
      IF(ISKEDTP-3)1441,1442,1441

C       BACKGROUND USERS ALLOWED

 1442 IQUE(IQ)=1
      IQ=IQ+1
      GO TO 1040
 1441 CONTINUE
 1440 IF(LDCHK)1040,1400,1040
 1400 IF(IQ-1)1040,1121,1040
 1121 IF(IT-1)1040,400,1040

C      1040 FOR ANY REQUESTS 400 FOR NC REQUESTS

 1040 IQ=IQ-1

C      STATISTICS GATHERING

      CLOCK=CLOCK+SOVRHD2
      SOVRHD=SOVRHD+SOVRHD2
      NQUE=IQ
      NOQ=0
      DO 1570 I=1,NU
      IF(STAT(I,9))1570,1570,1250
 1250 NOQ=NOQ+1
 1570 CONTINUE
      NOVLD=NOQ-NQUE-IT+1-LDCHK
      ITOVLD=ITOVLD+NOVLD
      TOVLD=ITOVLD
      AOVLD=TOVLD/CYCNT
      IF(MNOVLD-NOVLD)1380,1380,1390
 1380 MNOVLD=NOVLD
 1390 FMNOVLD=MNOVLD

C      EARLY TERMINATION

      IF(TERM)1530,1530,1540
```

```
 1540 FNQUE=NQUE-NEXT+1
      FNCYCTM=NCYCTM
      FNEXT=NEXT
      FNCYCTM=FNCYCTM-QAVE*FNEXT
      Q=FNCYCTM/FNQUE
      TERM=-1.0
      GO TO 1550

C     QUANTUM DETERMINATION FIRST PHASE

 1530 JQ=NCYCTM/NQUE
      Q=JQ
 1550 IF(IT-1)1500,1500,1030

C     QUEUE FORMATION SECOND PHASE

 1030 DO 1130 I=1,IT
 1130 IQUE(NQUE+IT-1)=IQUIT(IT-1)
 1500 IF(LDCHK)1520,1520,1510
 1510 IQUE(NQUE+IT)=ILOD
 1520 CONTINUE
      NWCYCL=1

C     QUANTUM DETERMINATION SECOND PHASE

      IF(Q-QA)1140,1150,1150
 1150 Q=QA
      GO TO 1020
 1140 IF(Q-QB)1160,1170,1170
 1170 Q=QB
      GO TO 1020
 1160 IF(Q-QC)1180,1190,1190
 1190 Q=QC
      GO TO 1020
 1180 IF(Q-QD)1200,1210,1210
 1210 Q=QD
      GO TO 1020
 1200 IF(Q-QE)1220,1230,1230
 1230 Q=QE
      GO TO 1020
 1220 Q=QF
      GO TO 1020

C     OLD CYCLE

 1020 NEXT=NEXT+1
      TERM=-1.0
```

```
      IF(NWCYCL)1410,1420,1410
C     STATISTICS GATHERING
 1410 CYCNT=CYCNT+1.0
      QTOT=QTOT+Q
      QAVE=QTOT/CYCNT
      TCYTM=TCYTM+CYTIME
      CYTMAVE=TCYTM/CYCNT
      FNQUE=NQUE
      TNQUE=TNQUE+FNQUE
      AVNQUE=TNQUE/CYCNT
      CSOVRHD=CSOVRHD+SOVRHD
      AOVRHD=CSOVRHD/CYCNT
      COMPEFF=(AVNQUE*QAVE/CYTMAVE)*100.0
      IF(QMAX-Q)1300,1300,1310
 1300 QMAX=Q
 1310 IF(CYTIMMX-CYTIME)1320,1320,1330
 1320 CYTIMMX=CYTIME
 1330 IF(MNQUE-NQUE)1340,1340,1350
 1340 MNQUE=NQUE
 1350 IF(SOVRHDM-SOVRHD)1360,1360,1370
 1360 SOVRHDM=SOVRHD
 1370 FMNQUE=MNQUE
 1420 CLOCK=CLOCK+SOVRHD1
      NWCYCL=0
      SOVRHD=SOVRHD+SOVRHD1
C     END OF CYCLE DETERMINATION
      IF(NEXT-NQUE-IT+1-LDCHK)2000,1240,1240
 1240 ENDCYCL=1.0
      LDCHK=0
      SOVRHD=0.0
      GO TO 2000
C  BASIC EXCHANGE PACKAGE
 2000 I = IQUE(NEXT)
      SWPOVD = 0.000300
      IF (STAT(I,2)) 2001,2001,2010
C IF GR 0 CHANNEL I NEEDS LOADING
 2010 CONTINUE
      CLOCK = CLOCK + EXOVD
      SOVD = SOVD + EXCVD
```

```
      MSIZE = GENR(I,6)
      IF(MSIZE + NDRUM - MAXDRM) 2011,2011,2012

C  NO LOAD

 2012 IF(MDRUM(I,3)) 2016,2013,2013
 2013 CONTINUE
      MDRUM(I,3) = -1
      SAVE(I,1) = CLOCK
      WAITC = WAITC + 1.0
 2016 NOLOAD = NOLOAD + 1
      GO TO 300

C LOAD OK

 2011 CONTINUE
      IF(MDRUM(I,3)) 2014,2015,2015
 2014 WAITT = WAITT + ( CLOCK - SAVE(I,1))
 2015 MDRUM(I,3) = MSIZE
      NDRUM = NDRUM + MSIZE
      STAT(I,2) = -1.0
      STAT(I,7) = 2.0
      STAT(I,8)=GENR(I,6)
      CLOCK = CLOCK + EXOVF
      SOVD = SOVD + EXOVF
      GO TO 200
 2001 CONTINUE
      IF (STAT(I,6)) 2002,2002,2020

C IF  GR 0  CHANNEL I IS QUITING

 2020 CONTINUE
      MSIZE = STAT(I,8)
      NDRUM = NDRUM - MSIZE
      CLOCK = CLOCK + EXOVO
      SOVD = SOVD + EXOVO
      DO 2021 IA = 1,9
 2021 STAT(I,IA) = -1.0
      GO TO 210

C NORMAL EXCHANGE

C EXCHANCE METHOD 1 NO CON

 2002 CONTINUE
      MSIZE=MDRUM(I,3)
      CORET = CORET + 1.0
```

```
      COREUT = COREUT + STAT(I,8)
      GO TO 2100
 2100 CONTINUE
      IF(MCORE(I,4)) 2151,2151,2150

C PROGRAM IS IN CORE

 2150 TLOAD = 0.0
      TDUMP = 0.0
      TEDUMP=0.0
      TELOAD= 0.0
      GO TO 2152
 2151 CONTINUE
      IF(LASTI) 2154,2154,2153
 2154 TDUMP = 0.0
      TEDUMP=0.0
      GO TO 2155

C EXCHANGE LAST USER

 2153 TDUMP = STAT(LASTI,8) * 0.000003
      TEDUMP = TDUMP
      STAT(LASTI,7) = 2.0
      MCORE(LASTI,4) = -1
      MDRUM(LASTI,4) = 1
 2155 TLOAD = STAT(I,8) * 0.000003
      TELOAD =  TLOAD
      STAT(I,7) = 1.0
      MCORE(I,4) = 1
      MDRUM(I,4) = -1

C ADVANCE CLOCK CORRESPONDING TO EXCHANGE TIMES

 2152 TEXCH = TEXCH + TDUMP + TLOAD + SWPOVD
      TEFEXCH = TEFEXCH + TELOAD + TEDUMP + SWPOVD
      SOVD = SOVD + SWPOVD
      SWPOVD = 0.0
      CLOCK=CLOCK + TLOAD + TDUMP
      LASTI = I
      GO TO  220

C     OUTPUT

  500 CONTINUE
      IF(WAITC) 559,559,560
  559 WAITC = 1.0
  560 CONTINUE
```

```
      AWAIT = WAITT/ WAITC
      COREF = COREUT/ CORET
      EXEFF = (TEXCH - TEFEXCH)/TEXCH
      ASWOVD=SOVD/CYCNT
      ASWAP=TEXCH/CYCNT
      DO 551 I=1,10
  551 DATA(I) = DIST(I)/DST
   11 TOTOUT(IR,1)=VARPARA
      TOTOUT(IR,2)=CYCNT
      TOTOUT(IR,3)=CYTMAVE
      TOTOUT(IR,4)=AVNQUE
      TOTOUT(IR,5)=QAVE
      TOTOUT(IR,6)=AOVRHD
      TOTOUT(IR,7)=COMPEFF
      TOTOUT(IR,8)=AOVLD
      TOTOUT(IR,9 )=QMAX
      TOTOUT(IR,10)=CYTIMMX
      TOTOUT(IR,11)=FMNQUE
      TOTOUT(IR,12)=SOVRHDM
      TOTOUT(IR,13)=FMNOVLD
      TOTOUT(IR,14)=NU
      TOTOUT(IR,15)   =   TEXCH
      TOTOUT(IR,16)   =   TEFEXCH
      TOTOUT(IR,17)   =   EXEFF
      TOTOUT(IR,18)   =   COREF
      TOTOUT(IR,19)   =   COREUT
      TOTOUT(IR,20)   =   CORET
      TOTOUT(IR,21)   =   AWAIT
      TOTOUT(IR,22)   =   NOLOAD
      TOTOUT(IR,23)   =   DST
      TOTOUT(IR,24)   =   SOVD
      TOTOUT(IR,25)   =   AVERAGE(1,6)
      TOTOUT(IR,26)   =   NU
      TOTOUT(IR,27)=ASWOVD
      TOTOUT(IR,28)=ASWAP
      TOTOUT(IR,29)=REQUEST
      DO 571 IM=1,10
      IN = IM + 30
      TOTOUT(IR,IN) = DATA(IM)
  571 CONTINUE
      IR=IR+1
  570 CONTINUE
      IF(ISTOPF )10,10,558
  558 CONTINUE
      IR=IR-1
      PRINT 51
      PRINT 60
```

```
      PRINT 88
      PRINT 89
      PRINT 54
      PRINT 55
      PRINT 80
      PRINT 81
      PRINT 82
      PRINT 515,(((TOTOUT(I,J),J=1,7),(TOTOUT(I,J),J=27,28)),I=1,IR)
      PRINT 52
      PRINT 83
      PRINT 84
      PRINT 85
      PRINT 516,(((TOTOUT(I,J),J=8,14),TOTOUT(I,29)),I=1,IR)
      PRINT 51
      PRINT 90, ((TOTOUT(IT,IS),IS=15,20),IT=1,IR)
      PRINT 91, ((TOTOUT(IT,IS),IS=21,26),IT=1,IR)
      PRINT 92,((TOTOUT(IT,IS),IS=31,40),IT=1,IR)
      STOP
      END

C     UNIFORM AND GAUSSIAN R. N. GENERATOR

      SUBROUTINE RANDOM
      DIMENSION RNU(12)
      COMMON GENR,AVERAGE,DIST,JN,JOBTYPE,CLOCK,DST,IOUTCON
      COMMON MAXDRM,MAXDSC,MAXCOR,IR
      COMMON GAUSRN, UNIRN
      ARN = UNIRN
      FIV = 1953125.*390625.
      DO 20 I=1,12
      IF(ARN)11,10,11
10    ARN=0.490843
11    Z=ARN*FIV
      Z1=Z*1.0E-10
      NZ=Z1
      FLOAT=NZ
      RNU(I)=Z1-FLOAT
20    ARN=RNU(I)
      GETRN=0.0
      DO 30 I=1,12
      CALC=RNU(I)+GETRN
30    GETRN=CALC
      GAUSRN=CALC/6.0
      UNIRN=RNU(12)
      RETURN
      END
```

```
C       JOB GENERATOR

        SUBROUTINE SET(JX)
        DIMENSION UNUM(6)
        DIMENSION GENR(50,10) , AVERAGE(10,8) , DIST( 10)
        COMMON GENR,AVERAGE,DIST,JN,JOBTYPE,CLOCK,DST,IOUTCON
        COMMON MAXDRM,MAXDSC,MAXCOR,IR
        COMMON GAUSRN, UNIRN
   51   FORMAT(1H1)
   60   FORMAT(/////////)
   63   FORMAT(//44X,20HFIGURE 6 (CONTINUED)//)
  615   FORMAT(I4,F10.1,2X,2I6,5F12.4,I9)
        I = 0
        J = JX
        JN = JN + 1
        DO 600 IC=1,5
        CALL RANDOM
        UNUM(IC) = GAUSRN
  600   CONTINUE
        CALL RANDOM
        RN = UNIRN
        TEMP = 0.0
        DST = DST + 1.0
  612   I = I + 1
        TEMP = TEMP + AVERAGE(I,7)
        IF(RN - TEMP) 610,611,611
  610   DIST(I) = DIST(I) + 1.0
        GO TO 614
  611   IF(I-10) 612,614,614
  614   CONTINUE
C ARRIVAL TIME
        CALL RANDOM
        GENR(J,1) = CLOCK + LOGF(UNIRN)*(-AVERAGE(I,1))
C LOAD
        GENR(J,2) = AVERAGE(I,2)*UNUM(1)
C ACTIVE TIME
        GENR(J,3)=AVERAGE(I,3)*UNUM(2)
C I/O TIME
        GENR(J,4)=AVERAGE(I,4) * UNUM(3)
C REPEATS
        GENR(J,5)=AVERAGE(I,5)*UNUM(4)
C SIZE
        GENR(J,6)=AVERAGE(I,6)*UNUM(5)
  620   CONTINUE
        MSIZE = GENR(J,6)
        IF(MSIZE-MAXCOR) 618,619,619
  619   GENR(J,6) = MAXCOR - 100
```

```
      GO TO 620
  618 CONTINUE
      IF(ICUTCON-4)616,621,616
  621 IF(IR-1)617,622,616
  622 JP=JP+1
      IF(JP-31)617,623,623
  623 JC=JC+1
      IF(JC-41)624,625,625
  624 IF(JC-1)617,627,617
  627 PRINT 51
      PRINT 60
      PRINT 63
      GO TO 617
  625 IF(JD)617,626,617
  626 JD=1
      PRINT 51
      PRINT 60
      PRINT 63
      GO TO 617
  617 PRINT 615,JN,CLOCK,JX,I,(GENR(J,K),K=1,5),MSIZE
  616 CONTINUE
      RETURN
      END
      END
```

```
      300.0        1.0        1.0        1.0        1.0     8000.0       0.25
      300.0        2.0        2.0        2.0        2.0     8000.0       0.25
      300.0        3.0        3.0        3.0        3.0    16000.0       0.25
      300.0        4.0        4.0        4.0        4.0    16000.0       0.25
      900.0
      900.0
      900.0
      900.0
      900.0
      900.0
400000 32000 32000
      0.200      0.150      0.100      0.075      0.050      0.025      0.050
      0.010      0.002     3600.0      0.1
  2   120 2 4
400000 32000 32000
      0.200      0.150      0.100      0.075      0.050      0.025      0.050
      0.010      0.002     3600.0      0.1
  2   220 2 4
400000 32000 32000
      0.200      0.150      0.100      0.075      0.050      0.025      0.050
      0.010      0.002     3600.0      0.1
 2 1 320 2 4
```